WebSphere® software

IBM

IBM WebSphere V5.0 Security Handbook

WebSphere Handbook series

WebSphere Application Server security in details

End-to-end security design with Patterns for e-business

Security integration with Tivoli Access Manager

This book is a draft document (Redpiece). It is mostly based on an early build (BETA3) of WebSphere V5; as the product develops the book gets updated. The book is expected to be published in its final stage when WebSphere V5 is released, and will be based on the final product.

Please note that the book has not been edited by ITSO editors. It has been published only to provide early information to the technical community. You will find grammar, style and formatting errors through the book, please ignore them and do not send comments on these. Comments and reviews are more than welcome, send an e-mail to the project leader Peter Kovari (peter.kovari@us.ibm.com).

Peter Kovari
Derek Carpenter
Paul Creswick
Floyd Langley
David Leigh
Rao Maheshwar
Piotr Kisielewicz
Stephen Pipes

Redbooks

ibm.com/redbooks

IBM

International Technical Support Organization

IBM WebSphere V5.0 Security Handbook
WebSphere Handbook series

August 2002

Take Note! Before using this information and the product it supports, be sure to read the general information in "Special notices" on page ix.

First Edition (August 2002)

This edition applies to V5 of WebSphere Application Server V5 Base Application Server and Network Deployment Package for use with the Red Hat Linux 7.2, AIX 4.3.3, AIX 5L, Windows 2000 Server.

This document created or updated on November 6, 2002.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 662
P.O. Box 12195
Research Triangle Park, NC 27709-2195

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Special notices

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

IBM trademarks

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

AIX 5L™	FAA®	Redbooks (logo)™
AIX®	IBM®	Redbooks™
Balance®	Lotus Notes®	SecureWay®
BookMaster®	Lotus®	SP™
DB2 Universal Database™	LPDA®	Tivoli®
DB2®	MQSeries®	WebSphere®
Domino™	Notes®	z/OS™
e (logo)®	Perform™	
Everyplace™	RACF®	

Other company trademarks

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

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

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

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

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

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

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

Preface

This IBM Redbook provides IT Architects, IT Specialists, application designers, application developers, application assemblers, application deployers and consultants with informaiton to design, develop and deploy secure e-business applications using WebSphere Application Server V5.

Part 1, "WebSphere security" on page 19 provides a detailed overview of WebSphere Application Server V5 Security. It starts up with J2EE security, goes into details with the modules and components of a J2EE enterprise application; you can also read about programmatic security techniques. The last chapter of this part shows all the security related administrtive items in WebSphere Application Server V5.

Part 2, "End-to-end security" on page 345 offers details about end-to-end security solutions where WebSphere Application Server V5 is part of an enterprise solution. You will find an introduction of Patterns for e-business, in which security is in focus. A very important chapter of this part will discuss the integration between WebSphere Application Server V5 and Tivoli Access Manager.

The "Appendixes" on page 439 provides additional information related to chapters from Part 1 and Part 2 and also describes the sample application available together with the book.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

The IBM Redbook team (left to right: Stephen Pipes, David Leigh, Piotr Kisielewicz, Rao Maheshwar, Paul Creswick, Peter Kovari)

Peter Kovari is a WebSphere Specialist at the International Technical Support Organization, Raleigh Center. He writes extensively about all areas of WebSphere. His areas of expertise include e-business, e-commerce, security, Internet technologies and mobile computing. Before joining the ITSO, he worked as an IT Specialist for IBM in Hungary.

Derek Carpenter has nearly 3 years with IBM, where he has been a member of the Developer Relations - Technical Services & Support (DR-TS&S) located in Dallas. Since joining IBM he has spanned his product knowledge by providing developer support for WebSphere Application Server, WebSphere Voice Server, WebSphere Studio Application Developer, WebSphere Studio, VisualAge for Java. Derek is currently working with Tivoli Security and Storage, and IBM Directory Services software platforms.

Paul Creswick is an infrastructure architect with Westpac Banking Corporation, Australia. He has worked in the design, development and implementation of several e-business applications utilizing WebSphere and Tivoli Access Manger including online Loan Originations Systems and Business Banking Portals. Currently he is implementing a security and network identity architecture to provide enterprise services utilizing Tivoli Access Manager.

Floyd Langley is an Advisory Software Engineer in IBM Developer Relations Technical Support. I currently provide support for IBM Tivoli Access Manager, and IBM Directory Server. I have a degree in Computer Science from the University of Kansas. I was in development in IBM on a variety of products for 13 years, and have been in technical support for the last 3. I currently hold certifications as a Microsoft MCSE - NT 4.0, IBM Certified Specialist - AIX 4.3 System Administration, Tivoli Certified Consultant - Tivoli Public Key Infrastructure V3.7.1, Tivoli Certified Consultant - IBM Tivoli Access Manager for eBusiness V3.9, and Tivoli Certified Consultant - IBM Tivoli Access Manager for Business Integration V3.8.1. My current areas of expertise are security and LDAP.

David Leigh is a Advisory Software Engineer in IBM Software Group's Websphere Platform System House organization, located in Research Triangle Park, North Carolina. He has six years of experience providing internal network application services and support. His areas of expertise include application and server security, high-availability, monitoring, problem determination, IBM AIX, and DCE/DFS.

Rao Maheshwar is a WebSphere Consultant with iS3C Consultancy Services Ltd, Inc. He worked as a J2EE programmer, designer, analyst and an architect for many web related projects. He did his engineering in Computer Science. He is experienced in Websphere Commerce Server, WebSphere Apps. server scalability, IBM DB2 UDB clustering using HACMP, WebSphere Edge Server for IBM HTTP Server load-balancing, WebSphere MQ and Web Services. Currently focussing on XML based web services and web security implementations.

Piotr Kisielewicz works as Advisory IT Specialist in IBM Global Services BIS Poland in e-business integration group. He is primarily responsible for architecting web based solution in the areas of integration and security. His expertise include web system design based on WebSphere and Domino as well as integration through various middleware technologies. Before joining IBM, he worked for a business partner as communication specialist. He holds an MSc degree in electronics from Technical University of Wroclaw and MBA from Ecole de Mines de Saint Etienne (France).

Stephen Pipes is a WebSphere consultant for IBM HS&T based in Hursley, England. He has several years of programming experience with Java and has worked for three years in the Java Technology Center, in Hursley before moving to the WebSphere development group. Stephen works with a number of customers providing technical support and education on a variety of WebSphere Application Server and Java topics.

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

International Technical Support Organization, Raleigh Center

Notice

This publication is intended to help IT Architects, IT Specialists, application designers, application developers, application assemblers, application deployers and consultants to design, develop and deploy secure e-business solutions using WebSphere. The information in this publication is not intended as the specification of any programming interfaces that are provided by WebSphere Application Server V5.0, Tivoli Access Manager 3.9. See the PUBLICATIONS section of the IBM Programming Announcement for WebSphere Application Server V5.0, Tivoli Access Manager 3.9 for more information about what publications are considered to be product documentation.

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other 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 Internet note to:

> `redbook@us.ibm.com`

► Mail your comments to the address on page ii.

1

Introduction

This chapter is a short introduction to the book. It gives a description of the organization of the book, and how to read the book according to your interest.

The book provides help to identify the chapters and sections to find, which J2EE role should read certain sections in the book.

1

1.1 How to read this book

There are really two approaches to discuss security for WebSphere:

► From the application point of view

► From the system point of view

If you are an application designer or developer then Part 1, "WebSphere security" on page 19 is for you, then you should get the big picture in Part 2, "End-to-end security" on page 345.

If you are a system architect and you want to calculate with security in advance then start with Part 2, "End-to-end security" on page 345, then you should read Part 1, "WebSphere security" on page 19 to see how the applications will work in the system.

In the book you will find the name WebSphere at numerous places. Although it is the name of a product family in this book WebSphere refers to the WebSphere Application Server product.

The development environment for the WebSphere product family, based on the Eclipse framework, is referred as WebSphere Studio in this book. Although WebSphere Studio has multiple different editions, the following editions can be used with this book, to accomplish the development tasks:

► WebSphere Studio Application Developer
► WebSphere Studio Application Developer Integration Edition
► WebSphere Studio Enterprise Developer

J2EE roles

Incorporating with Sun's J2EE roles defined in the J2EE Platform Specification, this book provides some additional information for those who would like to follow Sun's recommendations.

You will find icons on the side of the pages through the book supplementing section titles. Those icons are indicating the related J2EE roles for certian sections.
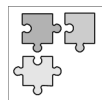
 This icon represents the Application Assembler role. The section noted with this icon provides information for application assemblers, for those who package the application from the components provided by the developers.
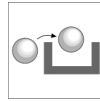
This icon represents the Deployer role. The section noted with this icon provides information for application deployers, for those who deploy the application(s) provided by the application developers and application assemblers.

This icon represents all the developer roles. The section noted with this icon provides information for developers. It is a collection of developer roles, not identifying any particular role.
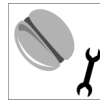
This icon represents the EJB developer role. The section noted with this icon provides information for EJB developers.

This icon represents the Java developer role. The section noted with this icon provides information for Java developers. It is really a collection of roles, because the majority of the developers in a J2EE environment are Java developers.

This icon represents the Web developer role. The section noted with this icon provides information for Web developers for those who develop Web pages, servlets, Java beans, access beans for EJBs and so on...
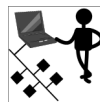
This icon represents the system administrator role. The section noted with this icon provides information for system administrators.

Indicating the roles for particular chapters and sections does not mean that others cannot read those parts in the book. Actually you are encouraged to read the whole book or any part that you are interested in.

The intention of the icons introduced above is to provide a mapping between the J2EE roles and the context of the book. It is quite difficult and does not really make sense to organize the book according to the J2EE roles, although the concept of J2EE roles is a good concept. Hopefully this approach will help to identify the tasks and to-dos for the reader, and adds more value to the book.

2

Security fundamentals

This chapter is a short introduction to security. It discusses security in a very general form, in order to have a common understanding on the topic.

The very basic security terms and definitions are covered in this chapter, independently from the rest of the book. Although you will not find any application server security or J2EE security in this chapter, it is still a good start and reference for later discussions.

2.1 Security

As new business practices emerge, most enterprises are finding that their existing security infrastructure is not capable of meeting the rapidly changing and more rigorous demands of business over the Internet. The demands of network security have now gone far beyond simply managing user accounts and restricting access between internal and external networks. These demands now require a sophisticated system that allows fine-grained access control to resources, yet is manageable enough to be tailored to protect systems from many types of security threats.

Security is a fairly vast topic; everything involves security to some extent, in a certain format. There are two main areas which have to be discussed separately:

► Physical security
► Logical security

Systems have to be protected both from outsiders and insiders. Do not forget that not every intrusion or attack is intentional; misuse of a system or improper administration can also cause damage.

2.1.1 Physical security

Physical security means protection against physical actions. It involves every physical element around:

► The machine(s) where the application is running.

► The room where the machines are operating.

► The building where the machines are installed.

► The site where the company is located.

The listed elements have to be secured against intrusion and damage, whether intentional or not.

Physical security also includes the protection of communication channels:

► Ground lines

► Wireless connection

The communication network has to be protected against eavesdropping and damage to the connection (cutting the line).

The subject of physical security goes much further than the objective of this book allows. This short section is only intended as a reminder of the concept of logical security.

2.1.2 Logical security

Logical security is related to particular IT solutions: the IT architecture and applications, including the business processes.

Communication

Network communication must be protected not only on a physical but on a logical level as well. Most of the companies' networks are connected to public networks. Therefore, applications are accessible from the outside world. Network level security must prevent unauthorized access.

Application

Securing an application is done on different levels. Security is designed from the very beginning of the implementation, when the processes and flows are designed.

► Securing the resources

This implies protecting the resources on an application level and exercising the security features of the runtime platform (authentication and authorization).

► Implementing the business processes securely

The processes have to be designed in a way that no weakness in logic can be found.

2.1.3 Security policy

Security policies are guidelines for an organization; they can be part of a widely accepted standard (ISO) or implemented by a certain organization or company.

Policies can define processes for different areas in an organization. Security policies focus on security related processes, for example, how to request a new password, how to renew a password, and so on.

These guidelines are very important in implementing a robust security for the whole system organization-wide.

2.2 Security fundamentals

This section will discuss two fundamental security services also supported by WebSphere Application Server:

► Authentication

▶   Authorization

2.2.1 Authentication

Authentication is the process of establishing whether a client is valid in a particular context. A client can be either an end user, a machine or an application.

> **Definition:** A *realm* is a collection of users that are controlled by the same authentication policy.

The authentication process involves gathering some unique information from the client.

There are three major groups of secure authentication used to gather this unique information:

▶   Knowledge-based - user name and password, for example.

▶   Key-based - physical keys, encryption keys, key cards.

▶   Biometric - finger prints, voice patterns or DNA.

Other authentication mechanisms can combine these; an example is digital certificates, where key-based and knowledge-based authentication are exercised.
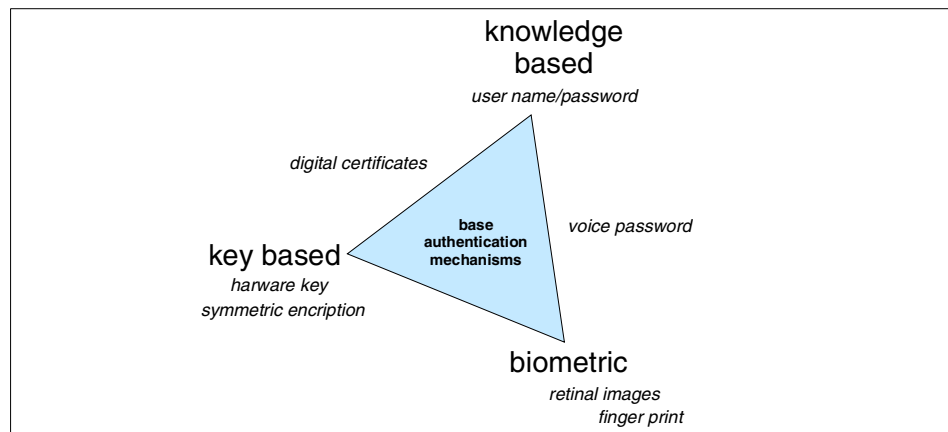
Figure 2-1   Base authentication mechanisms

The following paragraphs will discuss some of the authentication mechanisms used in IT systems.

User name and password

User name and password are the common method for authentication. The user who wants to access the system provides a user name and a password for login, which will be compared with the values stored in the system.

Physical keys

Physical keys are objects that can be used to prove the identity of the object holder. Physical keys can be a piece of metal used to unlock your computer, a hardware device that is plugged into the computer to execute certain programs or smart cards that have a memory or microprocessor embedded.

Biometric authentication

Biometric authentication is the use of physiological or behavioral characteristics used to verify the identity of an individual. The biometric authentication consists of comparing the physical characteristics of an individual against the values of those characteristics stored in a system.

Delegation

Delegation is the ability to leave an intermediary to do the work initiated by a client according to a delegation policy.

For example, in a distributed object environment, a client can request the method of an object on Server A. The method request results in invoking another method of an object in server B. Server A performs the authentication of the identity of the client and passes the request to server B. Server B assumes that the client identity has been verified by server A and responds to that request as shown in Figure 2-2.
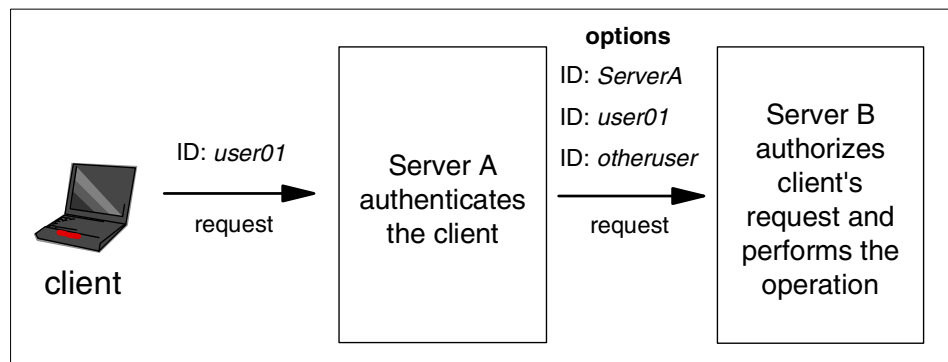
Figure 2-2   Delegation mechanism

Depending on the application environment, the intermediary can have one of the following identities when making a request to another server:

► Client identity: the identity under which the client is making the request to the intermediary.

► System identity: the identity of the intermediary server.

► Specified identity: identity specified through configuration.

2.2.2 Authorization

Authorization is the process of checking whether the authenticated user has access to the requested resource. There are two fundamental methods for authorization:

Access Control List

Each resource has associated with it a list of users and what each can do with the resource (for example: use, read, write, execute, delete or create).

Usually, an Access Control List specifies a set of roles allowed to use a particular resource and also designates the people allowed to play these roles.

For example, in a bank account object, we can have different methods (transfer, deposit, getBalance, setInterest, etc.). The access right can be granted on the basis of the roles of the users within the organization. A bank teller can have access to the *getBalance* method but not to *setBalance,* while a manager can access to both methods.

Table 2-1   Example of a Role Access Control List

Resources	Bank teller role	Manager role
getBalance method	yes	yes
setBalance method	no	yes

Capability list

Associated with each user is a list of resources and the corresponding privileges held for the user.

In this case, the holder is given the right to perform the operation on a particular resource.

In the previous example of the bank account object, the access right is granted to the user if the resource is listed in the user's capability list.

Table 2-2   Example of a capability list

Roles	getBalance method	setBalance method
Bank teller role	yes	no

Roles	getBalance method	setBalance method
Manager role	yes	yes

You will find the two tables shown above very similar, but the rows and the columns are switched. Actually, this is the difference between the two approaches. We have two sets: roles and resources. In the first case, roles are mapped to resources, while in the second case resources are mapped to roles.

The access control list is exercised generally, because managing security for certain resources is easier and more flexible than mapping resources to roles.

Role Based Security

Roles are different levels of security that relate to a specific application. For example, in a banking scenario, different employees have different roles, so the security access that each employee will require to complete the tasks in a web application will also be different. In a roles based authorization model, the roles for a given application are developed as an application is developed. As a user base for the application is established, one of three things happen.

► users are mapped directly to specifc security roles

► groups are formed, users are defined as members of a group, and the groups are defined to specific security roles

► a combination of user/group mapping to security roles is used to handle any exceptions

2.2.3 Public Key Infrastructure (PKI)

This section provides a brief overview of the Public Key Infrastructure (PKI). PKI is a part of IT security and today's security needs bring it into focus.

PKI is closely related to cryptography. Although it seems complicated, it is not. We do not need to use low-level mathematical algorithms, but we do need to understand the background involved.

Secret key cryptography

The secret key algorithms were invented earlier than were the public key algorithms. They use one key to encrypt and decrypt the data.
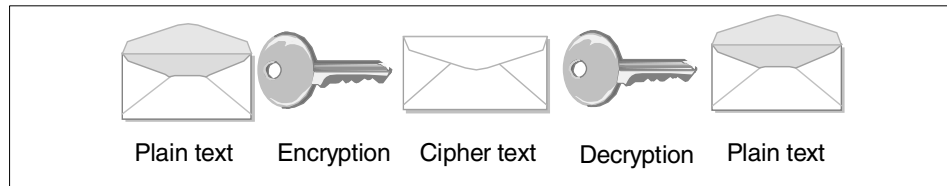
Figure 2-3   Symmetric key encryption

Figure 2-3 illustrates the concept of symmetric key cryptography. The algorithms used provide a great advantage: they are faster than the public key cryptography introduced later. They have a considerable disadvantage as well: the same key is needed for encryption and decryption, and both parties must have the same keys. In today's cryptography, the secret keys do not belong to persons but to communication sessions. At the beginning of a session, one of the parties creates a session key and delivers it to the other party; they can then communicate securely. At the end of the session, both parties delete the key and, if they want to communicate again, they must create another key.

The following section will discuss how to secure the delivery of the session key.

Public key cryptography

The first imperative of public key cryptography is the ability to deliver the session keys securely. It has many more benefits than secret key cryptography, as we will see in the following section.

Public key cryptography involves the use of different keys for encrypting and decrypting functions. If you encrypt something with key 1, you can only decrypt it with key 2, as shown in Figure 2-4.
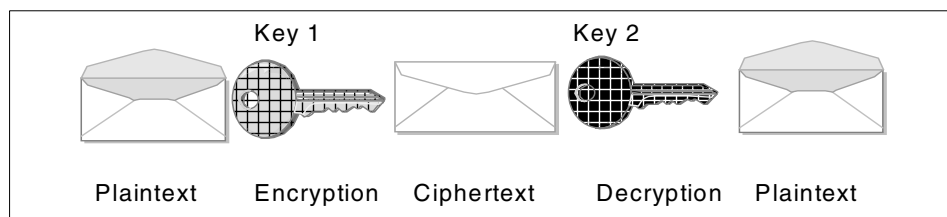
Figure 2-4   Public key concept

This architecture allows the use of one of the keys as a private key. This means that nobody can have access to this key except the owner. The other key can be used as a public key. If a user wants to send an encrypted message to another person, he or she will get the other person's public certificate, encrypt the message and send it. The message can be decrypted only by the owner of the private key.
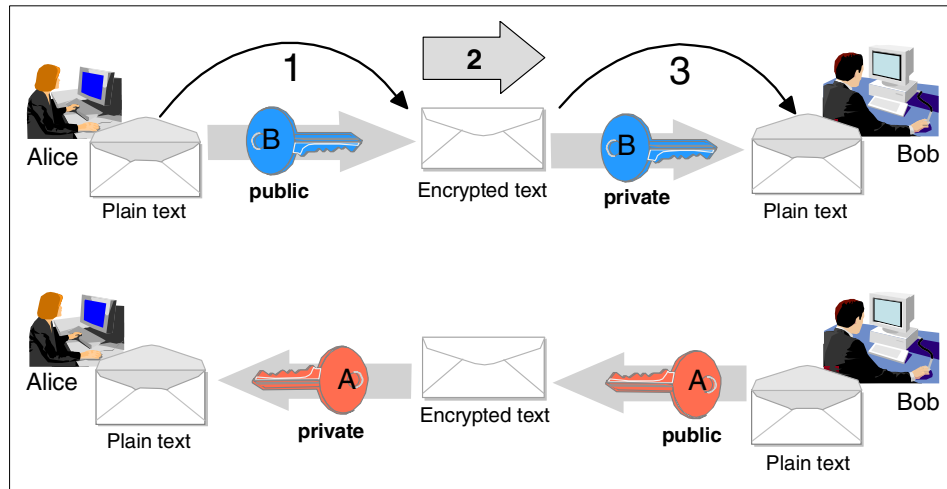
Figure 2-5   Using private key cryptography

Figure 2-5 shows a sample communication between two persons: Alice and Bob.

1. Alice wants to communicate with Bob but she does not want anybody to read the messages. She will use Bob's public key to encrypt the message.

2. Alice sends the message to Bob.

3. Bob uses his private key to decrypt the message.

If Bob wants to answer, he should use Alice's public key for encryption.

The example above is not suitable for the encryption of large amounts of data, because public key algorithms are very slow. We use the secure key algorithms to transmit large amounts of data. The session keys must be delivered with the public key algorithm and will be used during the communication.

It is the concept that SSL is following to estabilish a secure communication.

Certificates

A certificate is a document from a trusted party which proves the identity of a person. PKI certificates work in a similar fashion; if someone has a certificate from a trusted party, we can make sure of his or her identity.

Signatures

Signatures also work as in everyday life. Signatures used in the PKI environment work as follows: the information encrypted with a person's (the sender) private key will be unique to this person. Anybody can decode the message, and the source will be identified, because only one public key can open the message: the sender's public key. This message is almost good enough to be used for a digital signature; the only problem is that we would like to sign documents, and an encrypted document is too long to be a signature.

Signatures are not enough for identification. For example, if someone wants to travel by air, a passport will have to be shown as proof of identification.

The certificate, similar to a passport, is issued by a trusted authority. It should contain information about the owner and should be signed by the authority.

There is a standard defining the form of a certificate, called X.509. This standard also defines the attributes of a certificate, for example: X.500 name, issuer's name, distinguished name, serial number, and so on...

Elements of a certification authority system

A PKI system completes the tasks related to public key cryptography. These tasks should be separate, meaning that a PKI system should have some well-defined units to execute the different tasks. In some cases, the PKI implementation must separate the different functions physically (for example, in a commercial CA system). In this case , the elements listed below are located on different servers.

The logical elements of a PKI system are:

► Certificate Authority (CA)

► Registration Authority (RA)

► Certificate Repository (CR)

Certificate Authority (CA)

The CA component is the heart of a PKI system, it provides the "stamp" to the certificate. In some implementations, the CA component is issued together with the Registration Authority (RA) component. It stores its private key and can sign the certificate requests with it. This private key should be kept in a very secure place. If this key is corrupted, the whole certification tree will be unusable. It is possible to store this key on separate hardware.

Registration Authority (RA)

This component is responsible for the registration process. It is an optional component of a PKI system but, in most cases, it is implemented. The main RA task is the verification of client requests.

Certificate Repository (CR)

This component is often called a *certificate directory*. The users of a PKI system use the issued certificates to authenticate themselves. If someone receives a signed message, the receiver will check the signature. If the signature was issued by a trusted party the message will be considered a trusted message. Otherwise, there is a problem. The certificate could have been revoked for certain reasons (the owner left the company, owner's private key was corrupted, etc.). In this case, the certificate should not be considered to be a trusted one. This problem is solved by publishing certificates in the certificate repository. When a user receives a message with a certificate, the validity of the certificate can be verified.

The list of revoked certificates is called Certificate Revocation List (CRL) and is usually stored in the Certificate Repository (CR). The most common way of implementing a CR is to use the Lightweight Directory Access Protocol (LDAP) standard (RFC2587).

Certification process

Usually, there are two methods to issue certificates. The difference between the processes is the location where the client's private key will be generated.

1. In the first case, the client key pair is generated on the client side (on the client machine). The client will create a certificate request. The certificate request contains some information about the client (public key, name, e-mail address, key usage, some optional extensions, and so on). The request is signed with the private key of the client and sent to the server. The server identifies the client before issuing the certificate. The first step is to verify whether or not the signature at the end of the request is valid (the public key in the request can be used for validation). If no error is encountered, then either the certificate can be issued or another client validation process can be started. The most secure method of client validation is for the client to appear personally and certify themselves at the authority location. If the client certification is successful, the certificate for the public key is created with the desired key usage. The client can download the certificate into his/her browser registry or onto a smart card.

2. The other way to issue certificates is to execute the key generation process on the server side. This means that private keys should be created on the server side. This solution presents some problems:

   – The key generation requires a lot of computing power. There should be very powerful computers applied as Certificate Authority (CA) machines or key generation will be very slow (in case of multiple requests).

   – The private key must be issued and sent to the client, creating a weak point in the security.

There are situations when this method is better for issuing certificates. For example, let us imagine a research institute with a few hundred employees. The institute wants to make the entrance of the building more secure and also wants the computers to be used by the right persons. The company considers using smart cards for solving both problems. A PKI system can be implemented and every employee can get a smart card with a certificate and a private key. Obviously, the company will not establish a Web registration module for the employees (because of the fixed and small number of certificates to issue), but it will create the keys and certificates, install them on the cards and issue the cards to the customers. This process does not have any weak points, because the cards will be given personally to each proper person. Smart cards usually do not allow the exporting of private keys, so they cannot be corrupted (unless the card is stolen).

Infrastructure

A Public Key Infrastructure (PKI) system acts as a trusted third party authentication system. It issues digital certificates for the communication parties (for users and applications). Some of its tasks are:

► Issuing of certificates

► Revoking of certificates

► Renewal of certificates

► Suspension and resumption of certificates

► Management of issued certificates

► Issuing a list of revoked certificates

► Protection of the private key

The following diagram Figure 2-6 on page 17 shows three different certification scenarios on one picture.
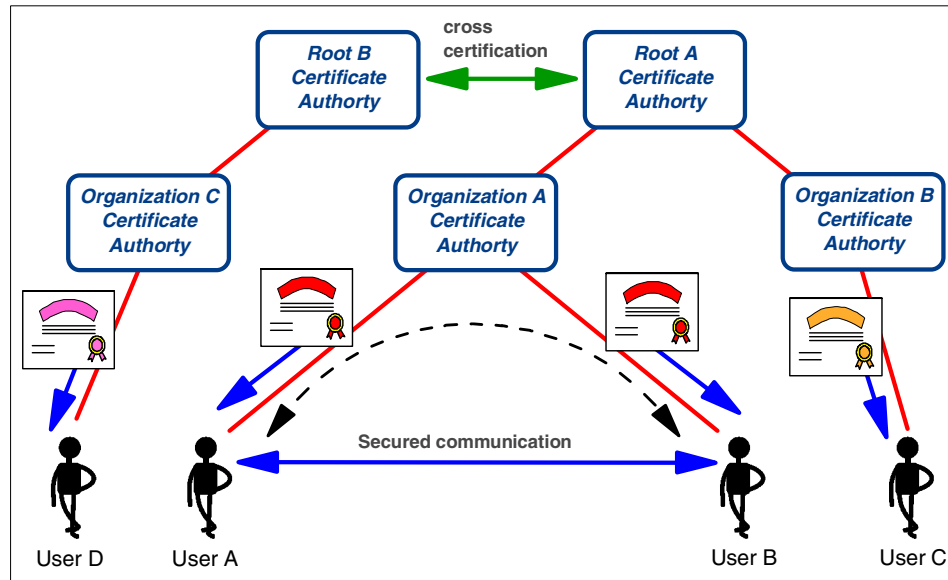
Figure 2-6   Simple certification scenarios

The certification scenarios depicted above:

► When User A wants to talk to User B, both of their certificates are issued and signed by the same Certificate Authority (Organization A), they can trust each other, the secure communication will build up based on the trust.

► When User A or User B wants to talk to User C, their certificates are coming from the same Root Certificate Authority (Root A), they can trust each other again. This scenario shows the hierarchy of the certificates, where the certificate has been signed by a chain of CAs. As long as the two parties have mutual Certificate Authorities along the line they can trust each other.

► When User D wants to talk to User A or User B or User C, their certification paths are different. To resolv the problem, the two root Certificate Authority (Root A, Root B) can set up a trust between each other, by setting up a cross certification. Once the two parties have cross certified CAs along the path, they can trust each other.

2.3 Security in use

Since security is a complex and diversified topic, it is important to keep it simple.

The following list will show the basic security areas. These areas have to be taken into account and their requirements must always be fulfilled.

► **Authentication / Identification** - Measures designed to protect against fraudulent transmission and imitative communications by establishing the validity of transmission, message, station or individual.

► **Access Control** - The prevention of improper use of a resource, including the use of a resource in an unauthorized manner.

► **Privacy / Confidentiality** - Assurance that information is not made available or disclosed to unauthorized individuals, entities, or processes.

► **Integrity** - The correctness of information, of the origin of the information, and of the functioning of the system that processes it.

► **Accountability / Non-repudiation** - Assurance that the actions of an entity may be traced uniquely to the entity. This ensures that there is information to prove ownership of the transaction.

► **Administration / Configuration** - Methods by which security policies are incorporated into the architecture and the functionality that the system architecture needs to support.

► **Assurance / Monitoring** - Confidence that an entity meets its security objectives; this is usually provided through an Intrusion Detection System.

► **Security Management** - Assurance that an entity meets its security management objectives, processes and procedures.

If you keep this list in mind during design and development, security will be well implemented.

Part 1

WebSphere security

19

3

J2EE application security

This chapter introduces the primary security aspects of J2EE platform, including:

► Introduction to *Security Roles*

► Discussion of the J2EE Container Based Security model

► How J2EE application security policies are administered in WebSphere during application assembly and during application deployment.

3.1 J2EE Application

The Java 2 Enterprise Edition (J2EE) specification defines the building blocks and elements of a J2EE application that builds an enterprise application. The specification also provides details on security related to the different elements.

The J2EE application consists of multiple modules, and components; these elements are in connection with each other, and they communicate via certain protocols. This section only discusses the connection on the application level, without going into details on protocols.

The purpose of the following diagram is to depict most of the elements in a J2EE application and their relation. You can find several "arrows" indicating connections between elements; these are the connections, connection groups, that have to be secured in a J2EE application.
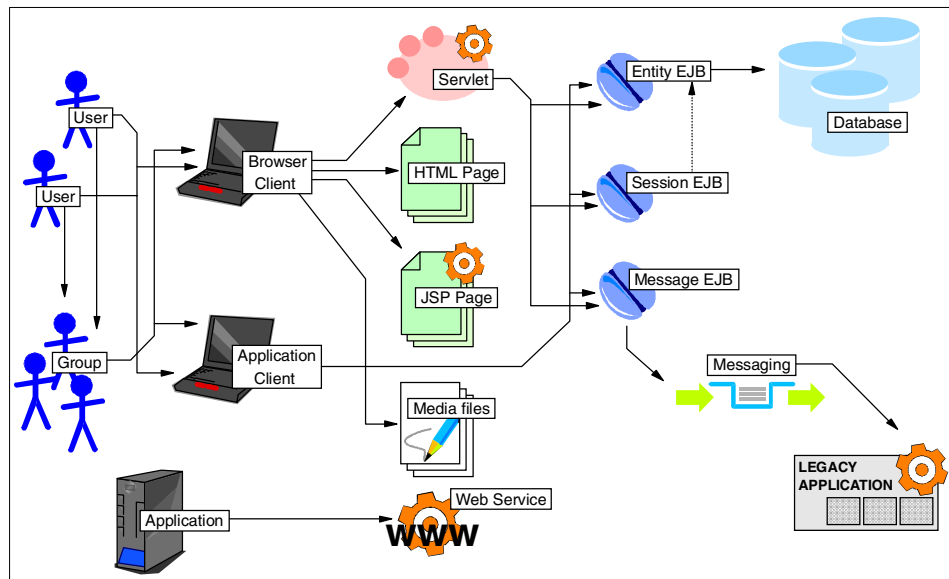
Figure 3-1   J2EE application

For example: the user accesses a JSP page on the application server; this JSP is a secured resource. In this situation the application server has to authenticate the user and decide whether the user is authorized to access the page or not. In this case the connection between the user's browser and the JSP page requires security.

In another example a servlet in the Web container on the application server accesses an EJB in the EJB container on the application server. Same thing happens like the previous example; the application server has to authenticate the servlet's request on behalf of the EJB; then check the authorization.

When you design an enterprise application or security for an application you will have a similar, but more detailed diagram for your solution. Make sure that you have taken every connection into consideration between each element and module. Security in this context consists of two major parts: authentication and authorization. Make sure that the access is always authenticated or the security credentials are propagated; also make sure that the access is authorized and prepare with an action if authorization is not granted.

For more information read the security related sections of the Java 2 Platform Specification v1.3 at `http://java.sun.com/j2ee/docs.html`.

3.2 Security Roles

The J2EE specification defines a security role as: *"A logical groupings of users that are defined by an Application Component Provider or Assembler".* Security roles provide a mechanism whereby application developers determine the security policies for an application by creating named sets of users (for example: managers, customers, employees, and so on) that will have access to secure resources and methods. At application assembly time, these sets of users, or security roles, are not tied to any real users or groups of users. Instead, they are placeholders which are later mapped to real users and groups at application deployment time, during a process called *security role mapping.*
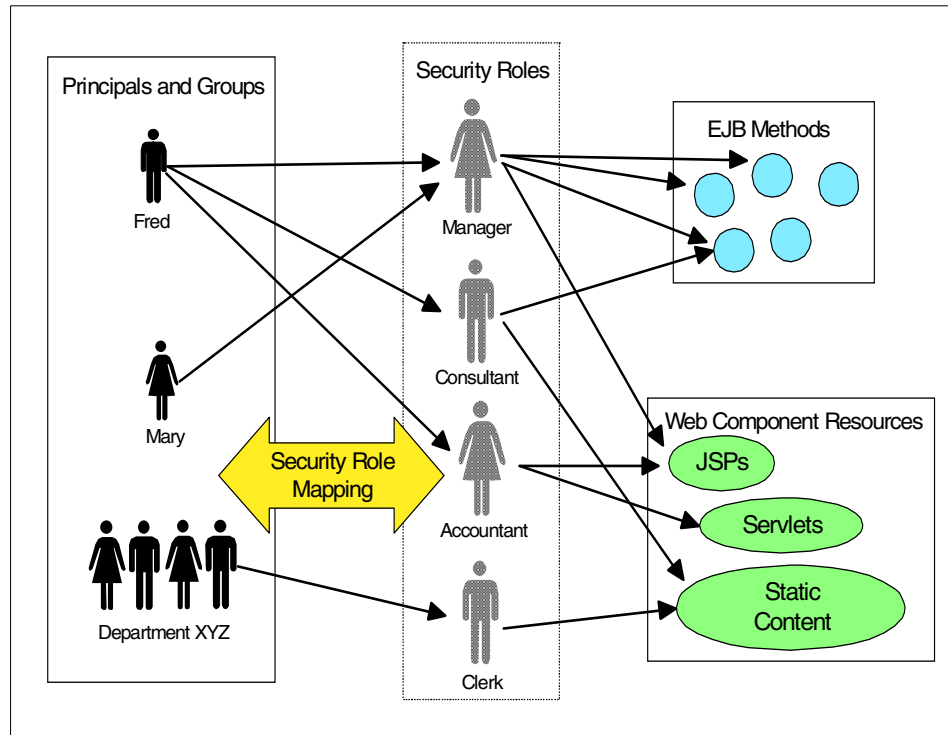
Figure 3-2   Security Roles

This two-phase security administration approach allows for a great deal of flexibility and portability. Deployers of an application have full control over how their local users and groups are mapped to the application's security roles, and over what authorization and authentication mechanisms are used to determine role membership.

At deployment time, security roles can be mapped to users, groups of users, or *Special Subjects*. There are two special subjects in WebSphere Version 5:

► All Authenticated Users

► Everyone

3.3 J2EE Container Based Security

J2EE Containers are responsible for enforcing access control on component objects and methods. Containers provide two types of security:

► Declarative Security

▶ Programatic Security

3.3.1 Declarative Security

Declarative security is the means by which an application's security policies can be expressed externally to the application code. At application assembly time, security policies are defined in an application's *deployment descriptor*. A deployment descriptor is an XML file which includes a representation of an application's security requirements, including the application's security roles, access control, and authentication requirements.

When using declarative security, application developers are free to write component methods that are completely unaware of security. By making changes to the deployment descriptor, an application's security environment can be radically changed without requiring any changes in application code.

3.3.2 Programatic Security

Programatic security is used when an application must be "security aware". For instance, a method might need to know the identity of the caller for logging purposes, or it might perform additional actions based on the caller's role. The J2EE Specification provides an API which includes methods for determining both the caller's identity and the caller's role.

The EJB methods are:

▶ `isCallerInRole`

▶ `getCallerPrincipal`

The HttpServlet methods are:

▶ `isUserInRole`

▶ `getUserPrincipal`

The use of these methods will be discussed in Chapter 8, "Programmatic security" on page 179.
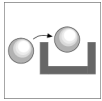
3.4 Application Deployment Descriptor

> **Note:** This section contains information about deployment descriptor elements which pertain to all J2EE components. Descriptions and examples of the deployment descriptor elements which pertain to specific J2EE Components can be found in Chapter 4, "Securing Web components" on page 37, and in Chapter 5, "Securing EJBs" on page 73.

There are two deployment descriptor files used for security role mapping:

Table 3-1   Role mappings in deployment descriptors

File Name	Purpose	Mandatory?
application.xml	Security Roles Defined	yes
ibm-application-bnd.xmi	Security Roles Mapped	No. Security Roles can be mapped during or after installation

In the application.xml file, all security roles used in the application must be named, with an optional description. The following example shows the XML elements required to define six security roles: manager, consultant, clerk, accountant, allauthenticated, and everyone:

Example 3-1   Security Role Definitons in application.xml

```
<security-role id="SecurityRole_1">
    <description>ITSOBank manager</description>
    <role-name>manager</role-name>
<security-role>
<security-role id="SecurityRole_2">
    <description>ITSOBank consultant</description>
    <role-name>consultant</role-name>
</security-role>
<security-role id="SecurityRole_3">
    <description>ITSOBank clerk</description>
    <role-name>clerk</role-name>
</security-role>
<security-role id="SecurityRole_4">
    <description>ITSOBank accountant</description>
    <role-name>accountant</role-name>
</security-role>
<security-role id="SecurityRole_5">
    <description>All authenticated users</description>
    <role-name>allauthenticated</role-name>
</security-role>
```

```
<security-role id="SecurityRole_6">
    <description></description>
    <role-name>everyone</role-name>
</security-role>
```

In the ibm-application-bnd.xmi file, security roles are mapped to users or groups in the User Registry. The next example shows how the security roles defined above would be mapped in the following way:

Table 3-2   Role mappings

Security Role	Mapped to
manager	managergrp
consultant	consultantgrp
clerk	clerkgrp
accountant	accountantgrp
allauthenticated	All Authenticated Users (special subject)
everyone	Everyone (special subject)

The following example is a code snippet from the ibm-application-bnd.xml file, that holds the binding information for the J2EE roles.

Example 3-2   Security Role Mappings in the ibm-application-bnd.xmi file

```
<authorizationTable xmi:id="AuthorizationTable_1">
    <authorizations xmi:id="RoleAssignment_1">
        <role href="META-INF/application.xml#SecurityRole_1"/>
        <groups xmi:id="Group_1" name="managergrp"/>
    </authorizations>
    <authorizations xmi:id="RoleAssignment_2">
        <role href="META-INF/application.xml#SecurityRole_2"/>
        <groups xmi:id="Group_2" name="consultantgrp"/>
    </authorizations>
    <authorizations xmi:id="RoleAssignment_3">
        <role href="META-INF/application.xml#SecurityRole_3"/>
        <groups xmi:id="Group_3" name="clerkgrp"/>
    </authorizations>
    <authorizations xmi:id="RoleAssignment_4">
        <role href="META-INF/application.xml#SecurityRole_4"/>
        <groups xmi:id="Group_4" name="accountantgrp"/>
    </authorizations>
    <authorizations xmi:id="RoleAssignment_5">
        <specialSubjects xmi:type="applicationbnd:AllAuthenticatedUsers"
xmi:id="AllAuthenticatedUsers_1" name="AllAuthenticatedUsers"/>
```

Chapter 3. J2EE application security   **27**

```
            <role href="META-INF/application.xml#SecurityRole_5"/>
        </authorizations>
        <authorizations xmi:id="RoleAssignment_6">
            <specialSubjects xmi:type="applicationbnd:Everyone" xmi:id="Everyone_1"
name="Everyone"/>
            <role href="META-INF/application.xml#SecurityRole_6"/>
        </authorizations>
</authorizationTable>
```

3.5 J2EE Application Security Configuration

There are two aspects of application security administration which apply to all
secured J2EE application components: Defining security roles (performed at
application assembly time), and Security role mapping (performed at deployment
time). Additional application security administration tasks which apply to specific
J2EE components will be discussed in later chapters.

Defining security roles can be performed using either of two WebSphere tools:

▶ Application Assembly Tool

▶ WebSphere Studio Application Developer

Security role mapping can be performed using either of the above tools, or can
be performed using the WebSphere Administrative Console as part of application
installation.

The following sections describe in detail how security roles are defined and
mapped using each of these tools.
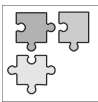
Defining Security Roles in the Application Assembly Tool

This section will show how to define J2EE roles on the application level. Normally
roles are defined in the individual modules and then collected automatically into
the application descriptor.

It is still useful to define security roles for the application, when the application
design and assembly follows the top-down design line or multiple assemblers are
putting together the application and there is a lead assembler who conducts the
assembly process. Security roles can be defined for the application and then can
be used on the module level, in this case the application will not end up using
different role names for the same role. Actually in the WebSphere Application
Assembly Tool you can copy and paste roles back and forth between the
application and its modules without creating them one-by-one.

The next steps will describe how to create the J2EE security roles for the application using the Application Assembly Tool.

1. Open the ear file, in the Application Assembly Tool.

2. Right-click the **Security Roles** item.

3. Select **New** from the pop-up menu.

4. A new window appears with the role details, fill out the fields according to the following screen capture.
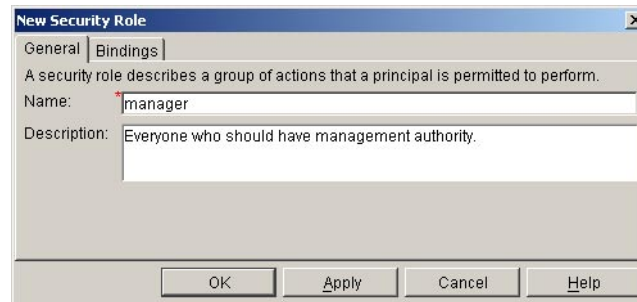
Figure 3-3   New J2EE role for the application

5. In case you want to do the role mapping you can switch to the Bindings tab and assign users, groups or special subjects to the role. It is not recommended to do the role mapping in application development or assembly time. Special subjects can be mapped to J2EE roles, they are static subject, they will not change on any platform or with any type of user registry.

6. Click **OK** to finish the form.

7. Create all the J2EE roles for your application repeating the steps above.

8. **Save** the .ear file.
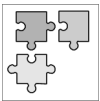
Security Role Mapping in the Application Assembly Tool

After a security role has been created in an EJB or Web module, the new security role will appear in the list of application-level security roles which can be seen by clicking on the application's **Security Roles** as shown in Figure 3-4.
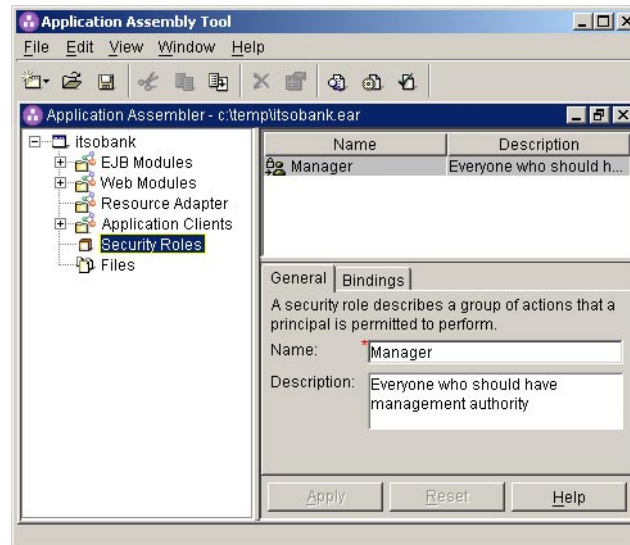
Figure 3-4   Application-level Security Roles

> **Note:** As security roles are added to EJB or Web modules, these new roles will appear in the list of application security roles, but they will *not* appear in other modules. This behavior is different from that in WebSphere Application Server Version 4. In Version 5, a module's security roles definitions are independent from other modules' definitions.

In the Application Assembly Tool, security role mappings are made in the Bindings tab in the application security roles view. To map the Managers security role to the Managergrp group, do the following:

1. Open the application-level **Security Roles** view (see Figure 3-4), and click the **Bindings** tab.

2. The **Bindings** tab contains fields for adding groups, users, and/or special subjects to a security role. Click the **Add...** button below the **Groups** heading to bring up the **Add Groups** dialog.

3. Enter the name of the real group, `Managergrp` and click **OK**.

4. The group mapping will now appear in the list of groups mapped to the manager security role.

Defining Security Roles in the WebSphere Studio

The method for adding security roles in the WebSphere Studio Application Developer differs depending on whether security roles are being added to the Web Module or the EJB module. See Chapter 4, "Securing Web components" on page 37, or Chapter 5, "Securing EJBs" on page 73 for details and examples.

Security Role Mapping in the WebSphere Studio

In the WebSphere Studio Application Developer,security role mapping is done as follows:

1. From the Resource Perspective, navigate to the application's deployment descriptor file, **application.xml,** and double-click this file.

2. Switch to the **Security** tab, and you should see a window similar to that in Figure 3-5.

3. Click the **Gather...** button to import all security roles that have been defined in EJB and Web modules into the list of security roles in the deployment descriptor.

4. Select the security role you wish to map, and check either one of the special subjects, **Everyone** or **All authenticated users**, or check **Users/Groups** to enter a user or group.

5. If entering a user or group, click the appropriate **Add...** button and enter the user or group name.
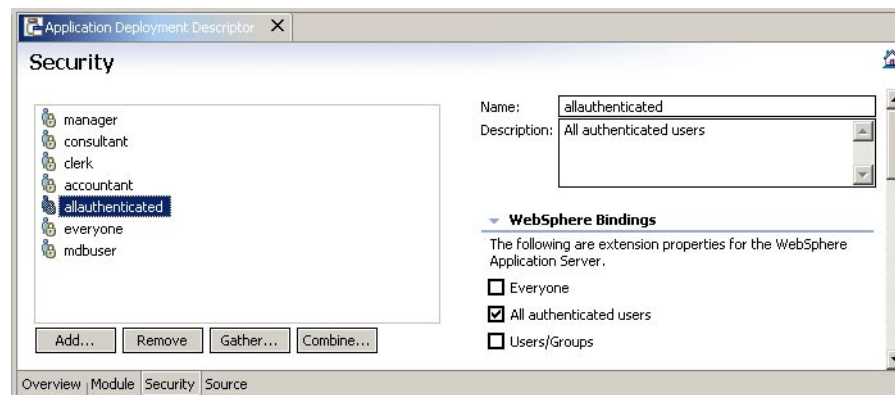
Figure 3-5   Security Role Mapping using the WebSphere Studio Application Developer
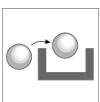
Security Role Mapping in the Administrative Console

Security roles mapped can be performed from the WebSphere Administrative console during application installation, or at any time once the applcation has been installed.

When installing an application using the WebSphere Administrative Console, one of the installation steps is to verify or define security role mapping. If security role mapping has been previously defined in the application's deployment descriptor, the console will display that mapping and allow it to be modified.

> **Note:** If no security roles are defined in the application deployment desciptor, this stop is omitted from the application installation steps.

After an application has been installed, the security role mapping console can be accessed by:

1. Click **Applications -> Manager Applications.**
2. Click the name of the application you wish to modify.
3. Under **Additional Properties**, click **Map Security Roles to Users/Groups.**

The Security Role mapping console appears as shown in Figure 3-6 on page 33. In this example, the manager role is mapped to managergrp, the clerk, consultant, accountant roles are also mapped to the according groups; the mdbuser role is mapped to a user: mdbuser; allauthenticated role is mapped to the All Authenticated special subject; and the everyone role is mapped the Everyone special subject.

> **Note:** Assign the special subjects, *All Authenticated* and *Everyone*, as the last settings before you click Next, then you will not lose these settings when you lookup users or groups.

Figure 3-6   Security Role mapping in the WebSphere Administrative Console

To map the *manager* role to the group managergrp in the LDAP user registry, do the following:

> **Note:** This example assumes that the user registry currently in use contains a group called: managergrp.

1. Check the box next to the manager role and click **Lookup groups.**

2. Enter `*managergrp*` into the search string field and click **Search.**

3. Select the group you wish to add to the role in the left-hand column, and click the right-arrow button "**>>**". The group will appear in the right-hand column as shown below.

    On the following screen capture, Figure 3-7, the application server is configured to use the LDAP user registry.
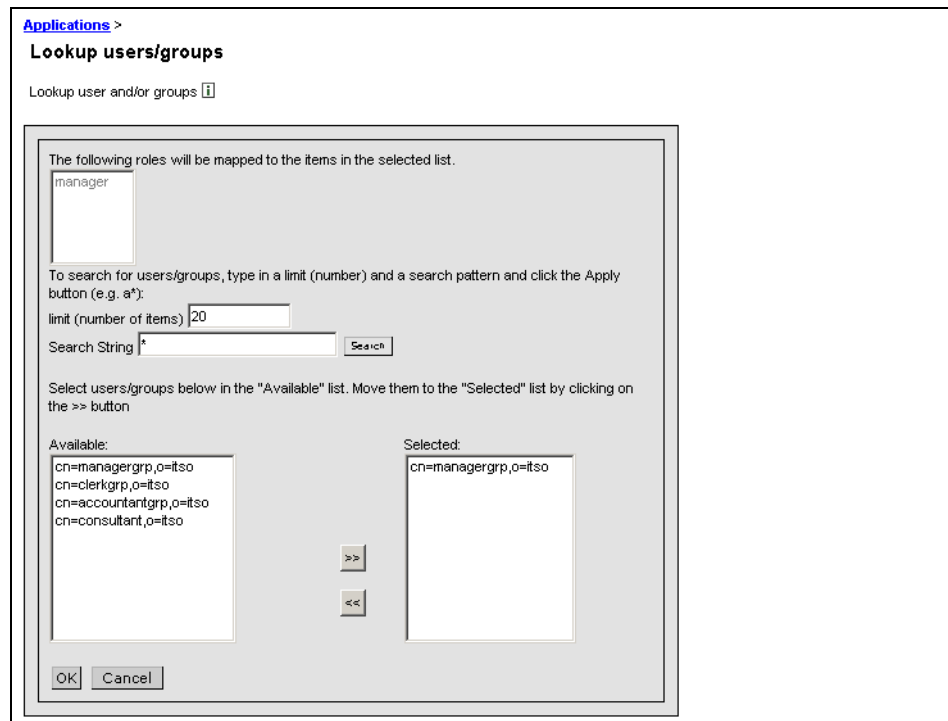
Figure 3-7   Searching for the manager group

4. Click **OK** to accept the changes and close the group lookup window

5. Click **OK** to accept and close the Security role mapping window.

6. Save the WebSphere configuration to make the changes effective.
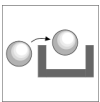
3.6 Modify applications

This section discusses two simple scenarios:

► Modifying the security role mapping for an enterprise application after deployment.

► Redeploying a secured application.

Modifying the security role mapping

There are certain situations when the role mapping needs to be changed after deployment or cannot be performed during deployment. WebSphere Application Server V5 lets the administrator to perform the sercurity role mapping on a deployed application using the Administrative Console.

If you want to edit the security role mapping for an application open the Administrative Console, and navigate to the **Applications -> Enterprise Applications**. Select the application, for example: `ITSOBank`, to get to the application details page. Select the **Map security roles to users/groups** link, which will bring up the same mapping page what you see during deployment.

You can do the security role mapping or you can change the actual mapping. After you are done with it, you have to restart only the enterprise application. The application will pick up the new mappings and will run with the new settings.

Redeploying an application

If you have a deployed and working enterprise application and you have to do some minor changes on the code or the application. This would be a case in a test environment or an application on a staging server. Redeploying the whole application every time you do some change is time consuming and unnecessary.

There is a fastpath for modifying and redeploying an application without loosing the deployment information. Here is how to do it:

1. Start the Administrative Console.

2. Select the enterprise application under **Applications -> Enterprise Applications**, then click **Export**.

3. Save the deployed application in a directory of your choice.

4. You can either modify the application using the Application Assembly Tool, or import it into WebSphere Studio and do the modifications there.

5. Once you are done with the modifications, go and redeploy the application in your WebSphere Application Server. During the deployment you will not have to do any new binding or mapping, unless you have changed something related to mapping and binding, just go to the last step, hit **Finish** and you are done.

4

Securing Web components

This chapter will discuss the security settings for Web components of the Web applications.

Web components such as static HTML pages, JSPs and servlets can be secured either by the HTTP server or by WebSphere.

In Section 4.1, "Static components" on page 38 we will discuss how to use the IBM HTTP Server mechanisms and settings to provide authentication and authorization for static pages.

In Section 4.2, "Web module security" on page 45 and following sections we will show how to secure static components belonging to WebSphere Application Server. We will demonstrate WebSphere Assembly Tool that comes with WebSphere.

4.1 Static components

WebSphere Application Server can only secure components that it owns. Any static pages that are served from the Web server can not be protected by WebSphere tools. They will require to use Web server related security mechanisms and will be transparent to WebSphere.

Most Web servers are able to secure the files that they serve. For example, IBMHTTP Server can protect its own resources, using:

► HTTP basic authentication uses user identity in the network or the user ID and password the user submits. The authentication can also be made based on combination of these elements.

► HTTP digest Authentication uses MD5 hash function to hash passwords and other data. Main idea of digest authentication is that the Web server does not store the users password in its authentication files but stores hashed (encoded) combination of strings that contain user ID, password and the authentication realm name.

► Digital certificate authentication using SSL uses SSL certificates to implement transport layer security for the TCP/IP protocol.

In Section 4.1.1, "Authentication with the Web server" on page 38, we provide an example of how to configure IBM HTTP Server to secure static content with HTTP basic authentication when user registry is stored in LDAP directory. In Section 4.1.2, "Authorization with the Web servera" on page 43, we explain how access to these static content can be managed using .htaccess configuration file.

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

External products may also be used to provide end-to-end security infrastructure. For information on how Tivoli Access Manage fits into this scenario see Chapter 12, "Tivoli Access Manager" on page 367.

4.1.1 Authentication with the Web server

It is possible to configure HTTP basic authentication for IBM HTTP Server (IHS) using the following user registries:

► Files: group names, user names and encrypted passwords are stored in files.

► Databases DBM and DB type database files are supported.

► LDAP directories: users and groups are defined in an LDAP server, such as IBM SecureWay Directory Server. This can be the same LDAP server that WebSphere uses for its user registry.

In this section will present simple scenario of how to implement basic HTTP authentication for the Web server when user registry is stored in LDAP directory.

To test the scenario we have used the following products:

► IBM HTTP Server, version 1.3.24, comes with WebSphere Application Server V5.0

► IBM Secure Way Directory Server, Version 3.2.2, comes with Tivoli Access Manager 3.9

► IBM DB2 v7.2 FP5

All products have been installed on a single server. In a case when LDAP server and database server are on a different machines client access software for DB2 and LDAP server will be necessary. The LDAP client software can be downloaded for from
`http://www-4.ibm.com/software/network/directory/downloads`.

We will enable security for all the static Web components in the C:\IBMHttpServer\htdocs\en_us directory.

The following instructions assume that all the software is installed and you already have an LDAP server populated with users. See Section 10.13.1, "IBM SecureWay Directory Server V3.2.2" on page 315 for details on how to do this.

The following steps will show you how to enable basic authentication for IBM HTTP Server.

Prepare necessary configuration files

The following steps will show you which files need to be defined for the Web server; and also how to use those files.

1. **ldap.prop** is an LDAP configuration file for the Web server. It is stored in the conf directory of the server (in our case it is `c:\IBMHttpServer\conf`). Sample LDAP configuration file with explanation of each directive is supplied with Web server software. For basic authentication the following entries are included.

Example 4-1   LDAP configuration for IBM HTTP Server

```
ldap.realm=LDAP Realm
ldap.URL=ldap://websrv01/o=itso
ldap.transport=TCP
ldap.application.authType=Basic
```

```
ldap.application.DN=cn=wasadmin,o=itso
ldap.application.password.stashFile=ldap.sth
ldap.user.authType=Basic
ldap.group.name.filter=(&(cn=%v1)(|(objectclass=groupofnames)(objectclass=group
ofuniquenames)))
ldap.group.memberAttributes=member uniquemember
ldap.idleConnection.timeout=600
ldap.waitToRetryConnection.interval=300
ldap.search.timeout=10
ldap.cache.timeout=600
```

where,

- – **ldap.URL** is of the form `ldap://<hostName>/<BaseDN>`
- – **ldap.application.DN** is the DN by which the Web server authenticates itself to the LDAP Server.

2. **ldap.sth** is a stash file containing encrypted password for the Web server to authenticate with LDAP. You need to decide with which user name and password Web server will connect to LDAP. To create the stash file enter at the command prompt:

```
C:\IBMHTTPServer\ldapstash <password> C:\Program
Files\IBMHTTPServer\ldap.sth
```

Configure your Web Server to use LDAP for authentication

The following steps will describe how to configure the IBM HTTP Server to use LDAP for authentication.

1. Add the LDAP module to the Web server configuration. From a Web browser, go to the IHS server configuration. Go to http://localhost then select **Configure server**. When prompted, enter your Web server administration ID and password.

> **Note:** If you have not set up an administration ID, then from a command prompt, enter:
>
> ```
> C:\Program Files\IBMHTTPServer\htpasswd -c C:\Program Files\IBM
> HTTPServer\conf\admin.passwd <adminName>
> ```
>
> When prompted, enter your chosen password and verify it. Stop and restart both Web server services to make the changes effective.

2. At the IBM Administration Server Interface select **Basic Settings -> Module Sequence**. Make sure that the Scope is set to **<GLOBAL>**. Click the **ADD** button.

3. From the **Select module to add** drop-down list select **ibm_ldap (IBMModuleLDAP.dll)** and click **Apply** button.

4. Click **Submit** button. You should see **ibm_ldap** module added in the Prioritezed list of active server modules window, as in the picture below.
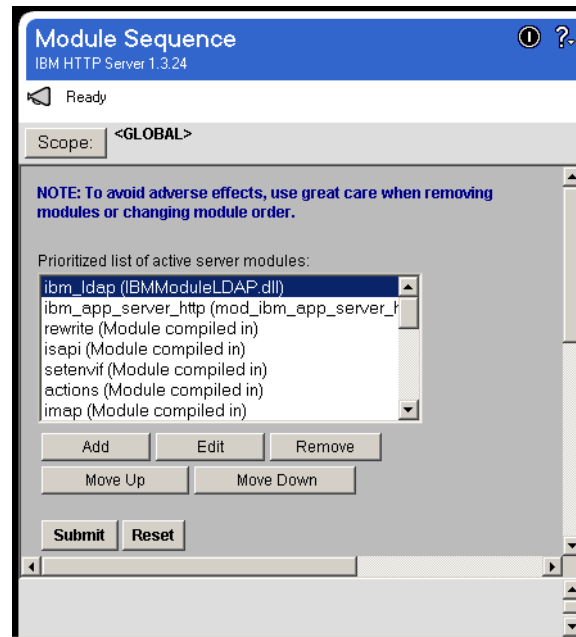
Figure 4-1   Adding LDAP server module into HTTP server configuration

5. Next, set the scope and type of the authentication. As mentioned before all documents that are under htdocs\en_us will be secured. From the Web server administration console, select **Access Permissions -> General Access.** Click the **Scope** button and select **<Directory C:\IBMHttpServer\htdocs\en_us>**.
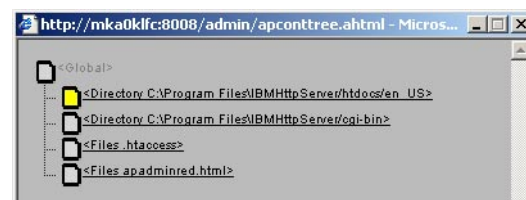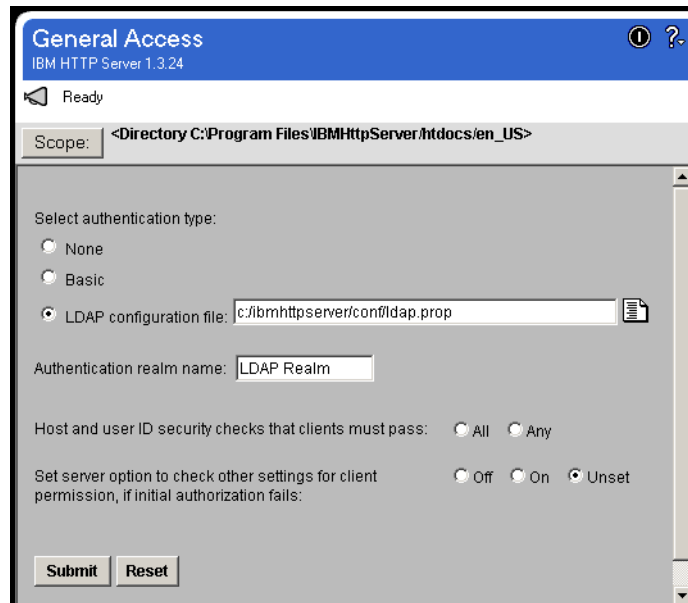
Figure 4-2   Defining authentication scope for LDAP basic authentication

6. Now select **LDAP** as the authentication type and enter the name of the configuration file you created in step 1 above,

`c:/IBMHttpServer/conf/ldap.prop`. Enter an authentication realm name; we used **LDAP Realm**.

Figure 4-3   Finalizing LDAP Configuration

7. Click **Submit.**

8. Close your browser from which you were administering the server. When you start your browser again and go to http://<hostname>, which in our case was `http://localhost`, you should be prompted with HTTP basic authentication window. Enter valid user name and password to be allowed to view the index page.

> **Note:** Configuration changes you have made through the Web Server Administration Interface added the following lines in the httpd.conf file in the section <Directory "C:/IBMHttpServer/htdocs/en_US">**:**
>
> ```
> ConfigFile "c:/ibmhttpserver/conf/ldap.prop"
> AuthName LDAP Realm
> AuthType basic
> Require valid-user
> ```

4.1.2 Authorization with the Web server[a]

By default the Web server configuration and access control directives are handled by Web server administrator by modifying the httpd.conf file. The appropriate section of the file enforces these settings.

Example 4-2   Enforcing access control management by settings in httpd.conf file

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

Directive *AllowOverride None* tells the Web Server not to look for any other access control definition files within 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 may require administrator's intervention in case of simple changes to the file. Second, 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 need to set an access control on per-directory basis, overriding the settings in httpd.conf file, IBM HTTP Server uses *.htaccess* files for every directory over which the user wants to have such control.

The use of *.htaccess* files adds possibility to dynamically configure security components for static portion of the Web site. Changes done to any *.htaccess* file do not require restart of the Web server or any other administrator's intervention since the file is read every time every time a resource is fetched from that directory.

A *.htaccess* file placed in one directory applies to all its subdirectories. In such a case it is equivalent to a *<Directory>* section in httpd.conf file. If there is more than one access files in a directory tree, the directives set in a file for subdirectory take precedence over the directives in the parent directory.

There is a number of directives that can overridden. When dealing with security we are interested in *AuthConfig* category of directives that will allow the use of authorization directives such as *AuthUserFile*, *AuthGroupFile*, *AuthDBMGroupFile* and others. To override this category, change the directive in *httpd.conf* file to:

Example 4-3   Security configuration in httpd.conf

```
<Directory "C:/IBMHttpServer/htdocs/en_US">
```

```
        AllowOverride AuthConfig
        Options None
</Directory>
```

For more information on how to use .htaccess see the Apache tutorial at
http://apache-server.com/tutorials/ATusing-htaccess.html

The drawback of using *.htaccess* files is a negative impact on a performance of
the Web server. As mentioned before, when the use of .htaccess files is enabled,
for any resource requested from one directory, Web server also checks all parent
directories for .htaccess files and tries to merge the configuration in order to
decide whether the user is entitled to read the resource or not.

The other problem with the .htaccess files is the system management. It is
difficult to maintain them especially in a centralized security infrastructure.

4.1.3 Other Web server security aspects

This book is not decicated fully to discuss Web server security, and only sample
scenario has been chosen to show how to secure Web server uisng LDAP user
registry. This LDAP server can be shared with applicaion server in order to
manage user's authorisation to specific Web application resources. This is
discussed in the following chapters.

However when writing about Web server security it not necessary to mention
other possible means that can be used to secure either Web server daemon
process or user access to Web server resources. Below is a short list of what you
can take into account for securing the Web server when designing the system to
run secure Web applications:

► Runnig daemon as root: when the HTTPD daemon starts it uses root account.
  Then it initilizes a number of threads that actually serves end-user requests.
  These threads are managed by the root's controller thread and are not
  responsible for administration and control procedures. The StartServer
  directive of the httpd.conf file sets the users and groups that own running Web
  server process. You have to set these directives to the users and groups that
  you define to run a Web server. Never run Web server as root user.

► Digest authetiction: basic authentication does not scure user passwords that
  are passed from Web browser to the server in plain text. In order to encypt or
  encode password some servers have additional modules that can implement
  MD5 encoding. Both your server and your browser should support MD5
  digest authentication. In digest authentication password is not send acrooss
  the network. On a client side, the Web browser creates a MD5 encoded string
  using user password and AuthRealm. Web server creates its own string

based on the information stored in httpd.conf file and compares it with the infomrmation sent by the client's browser.

► Kerberos authetication: Kerberos is third party authetication system which allows secure authentication and communication of clients and servers over the network. It uses DES algoritm for encryption. Refer to your Web server documentation for required modules in order to run Kerberos system.

► Chroot: many operating system offers chroot command that tells the application to treat a given directory as it would have been a root directory. This allows to hide the file system that is above that directory from every process of the executing application. One drawback of using chroot command is that all the executable code and modules used by the application should be placed within the directory that is visible to the application.

There is many other possible security option that may be used in your solution. Everything is higly dependent on the selected architecture and application requirements.

4.2 Web module security
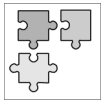
In a J2EE application architecture Web module of the enterprise application is comprised of one or more related servlets, Java Server Pages (JSP files), XML and HTML files that can be managed as a one integrated unit. The files in Web module are related together in a sense that they perform a common business logic function.
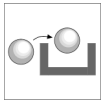
The Web modules of the enterprise application running within the Web container of the application server. Web container, as a run time environment for the Web application, is responsible for handling requests for servlets, JSP files and other components running at the server-side. The Web container creates servlet instances, loads and unloads servlets, creates and manages requests and response objects and performs other servlet management tasks. The Web server plug-in provided by the WebSphere Application server is responsible to redirect the client's request to the application server.

This section describes the process and tools of WebSphere Application Server to configure security for the Web module of enterprise application.

4.2.1 Configuring Web module security

One of the tools used to configure security settings for a Web module is the Application Assembly Tool (AAT).

Authentication method

The authentication method defines how the user will be authenticated by the Web application. Before any authorization constraint will be applied the user will need to pass authentication process using configured mechanism. The possible options are:

► **Basic authentication**

The user name and password are encoded by the browser and included in the HTTP request. This mechanism does not provide server authentication. The Web server sends to the client a request containing realm name in which the user will be authenticated.

► **Client certificate authentication**

The client certificate is transported across an SSL secured connection to the Web server. The Web server then extracts the credentials from the certificate and forwards them to WebSphere along with the request.

► **Form-based authentication**

This method allows the developer to control the authentication process. By default, the values that the end user supplies in the form are transmitted in clear text as parameter values in the HTTP request. To secure the user information during transmission, the connection should be encrypted.

The Application Assembly Tool has an option for *digest authentication*, but this option is not supported by WebSphere at this moment. If a security constraint has been set but no authentication method for a Web module has been configured, the default is to use basic authentication.

To set up an authentication method for Web application:

1. Load your Web application module into Application Assembly Tool in our example: itsobank.ear.

2. Click **itsobank -> Web Modules** to expand the tree.

3. Right-click the **itsobankWeb** Module and from the pop-up menu select **Properties.**

4. Select the **Advanced** tab.

5. Check the **Login Configuration** check-box, select appropriate authentication method and provide the Realm name that will be used by the Web server during the authentication.

6. Click **OK** to approve the changes.
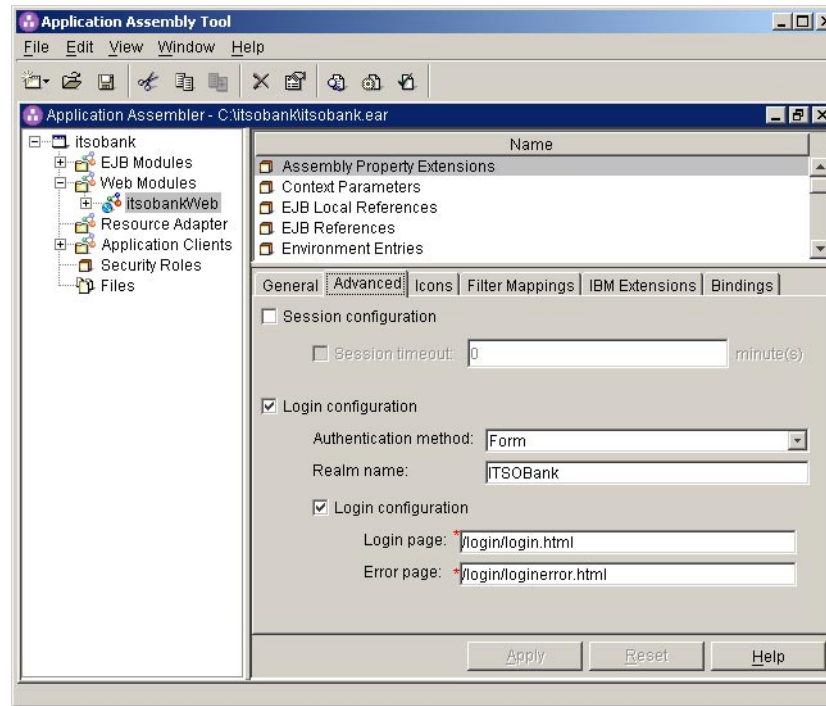
Figure 4-4   Login Method Configuration for itsobank application

Security roles

WebSphere implements the roles based security from the J2EE specification. Security role is a logical grouping of princpals. Access to a specific part of the application is granted based on the role, which is then mapped during development or deployment phase to specific user registry entries. It gives a certain level of transparency for the application development process. The developer needs not to bother about the different user privileges that can be defined for the application.

The security roles for the ITSOBank application were defined in Appendix A, "Sample application" on page 441. As noted there, security roles defined for a Web module are visible at the Enterprise application level. The following steps will describe how to define a role for the Web module with the Application Assembly Tool.

1. Open the itsobank.ear file in the Application Assembly Tool.

2. Right-click the **itsobank -> Web Modules -> itsobankWeb -> Security Roles** item.

3. Select **New** from the pop-up menu.

4. Fill out the fields according to the following screen capture.
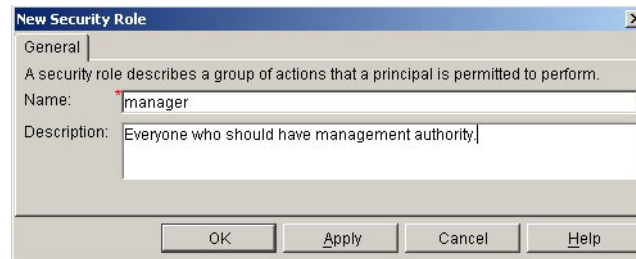
Figure 4-5   New role for the Web module

5. Click **OK**.

6. Repeat the steps above to create all the necessary roles for the Web module.

7. Save the .ear file.

Defining security constraints

Providing an authentication mechanism for global application security does not provide the mechnisms to control access to the Web resources.

Security constraints declare how the content of the application is protected. For a given security constraint three things should be defined:

► One or more Web resources that define actual application components that are to be protected by the security constraint. Web resource is a set of URL patterns and HTTP methods in those resources. All requests that that will be matced with the pattern defined for a given Web resource will be subject to a security constraint.

► Authorization constraint that defines roles which will be provided access to the Web resources existing within security constraint. An authorization constraint is a set of roles that the user must be granted in order to have access to a Web resource collection existing wiithin a security constraint. In order to have access to the Web resource user shoud be granted at least one of the roles that are defined within the Authorization constraint.

► Used Data Constraint idicates the transport layer setting for client/server communication in order to satisfy given security constraint. This settings should guarenatee either content integrity (preventing tampering in transit) or confidentiality (preventing reading data during transfer). User data constraint may override standard security settings for the application. For example access to some functions of the application may require just basic login using user ID and password, and in the same time some functions may require

higher level of protection. User Data Constraint allows an applicaiton deployer to introduce such protection.

If global security is enabled, and a security constraint is set for a particular resource, then the resource is secured.
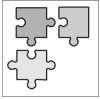
4.3 Securing Web components

This chapter presents simple scenarios of how to secure different Web components. We assume that that the roles are defined for the application as in the ITSOBank sample application.

4.3.1 Static content

Static resources of the enterprise application can be secured only if they are served by WebSphere. WebSphere can not manage access to the static content that resides on Web server. All the static content that needs to be protected by WebSphere Application Server must be packaged into the Web module (.war, Web Archive file). Static HTML pages can be served by the servlet that implements file serving behaviour.

Following instructions shows how to set up security constraint to protect static content for the Web application module using the Application Assembly Tool. This section will only provide information for the the Application Assembly Tool, since securing static contents within WebSphere does not differ from securing dynamic content in WebSphere, the book will provide information for WebSphere studio in the next section where dynamic components will be secured.

1. Start Application Assembly Tool and load the ITSOBank sample enterprise application archive file (itsobank.ear).

2. Expand **itsobank -> Web Modules -> itsobankWeb** and select **Security Constraints.**

3. Right-click **Security Constraints** and select **New**, A dialog window **New Security Constraint** will be opened.

4. Enter the security constraint name `Constraints for bank access`. Click **Add** next to the Roles area. You will be presented with with a dialog box listing the security roles that are defined for your application.

5. Select **Everyone** and click **OK**. You should be presented with the window like the one below.

Figure 4-6   Completed New Security Constraint Window

Note: User Data Constraint of this window allows you to chose a *Transport guarantee*, which defines how the communication between the client and the server is to be protected. There are three options to choose from:

► **None**

No constraint indicates that the application does not require any transport guarantee.

► **Integral**

This ensures that data cannot be changed in transit. In practice, this means that a request must be transmitted over an SSL encrypted channel.

► **Confidential**

This ensures that data cannot be viewed in transit. In practice, this means that the request must be transmitted over an SSL encrypted channel. See "Configuring the Web Server to support HTTPS" on page 239 for more information on configuring SSL.

4.3.2 Servlets, JSPs

WebSphere Application Server can secure dynamic resources such as servlets using role based declarative security mechanisms. This means that logical application security structure is defined independently from the application itself. The logical security structure is stored in deployment descriptors of the application.

For servlets WebSphere Application Server allows to protect the resources on the 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:

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

Using method level security constraints for servlets you may want to separate dynamic content that all the users can view from the administrative functions that only privileged users will be allowed to access. In WebSphere Application Server this is done using different security constraints for the different servlet methods.
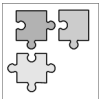
Configuring Security Constraints with the Application Assembly Tool

The following steps will show, how to define security constraints with the Application Assembly Tool.

1. Load the **itsobank.ear** application file into the Application Assembly Tool.

2. Expand the **itsobank -> Web Modules -> itsobankWeb** and select **Security Constraints**. Right click **Security Constraint** and select **New**. You will be presented with the **New Security Constraint** window.

3. In the Authorization Constraints section panel, next to Roles, click **Add**. You will be presented with a dialog box listing all the security roles that are defined for your Web module of ITSOBank application. Select the **clerk** and the **Manager** role and click **OK**.

4. Click **OK** to save security constraint. You should see your new security constraint listed in a **Constraints** panel.

5. In the left-hand pane, expand the new constraint and select **Web Resource Collections**. Right-click **Web Resource Collections** and select **New**. You will be presented with the **New Web Resource Collection** entry window.

6. Enter the Web Resource Name: **Customer transfer**.

7. Next to HTTP Methods click **Add**. You will be presented with the **Add HTTP methods** dialog box.

8. Add the **GET** and **POST** HTTP methods to the list.

9. Next to URLs , click **Add**. You will be presented with the **Add URLs entry** box.

10. Add the following URL patterns:

```
/transfer/customertransfer.html
/transfer/transferresults.jsp
/serlvets/Transfer
```

Click **OK**.

Final result of the resource collection for the security constraint definition is presented on the picture below.

Figure 4-7   Security constraints resource collection definition

Securing components using WebSphere Studio

The approach to define security constraints in WebSphere Studio Application Developer is very similar to the one described above. In order to peform the same task for Transfer servlet security constraint, follow the steps:

1. Open or select the J2EE perspective, and switch to the J2EE Hierarchy view.

2. Open the **Web Modules** folder, then double-click the **itsobankWeb** item. This will open the editor with the Web deployment descriptor. Switch to the **Security** tab.

3. Define a new security constraint, select **Security Constraints** tab at the top of the editor, then click the **Add** button under the **Security Constraints** window list-box; a new item appears in the box, and one under the **Web Resource collections** section.

4. Select the **(New Web Resource Collection)** in a **Web Resource Collections** section and click **Edit** button. This should open **Web Resource Collection** dialog box.

5. In the **Name** field enter `customer transfer.`

6. In the **HTTP Methods** window check the boxes for the **POST** and **GET** methods.

7. In **URL patterns** window type the following entries:

```
/transfer/customertransfer.html
/transfer/transferresults.jsp
/serlvets/Transfer
```

   Click **OK**, this should update the entry in Web Resource Collection section.

8. In the **Athorized Roles** section click the **Edit** button, the Select Auth Constraints dialog box should be opened.

9. Check the box for **manager** and **clerk** role and click the **OK** button.
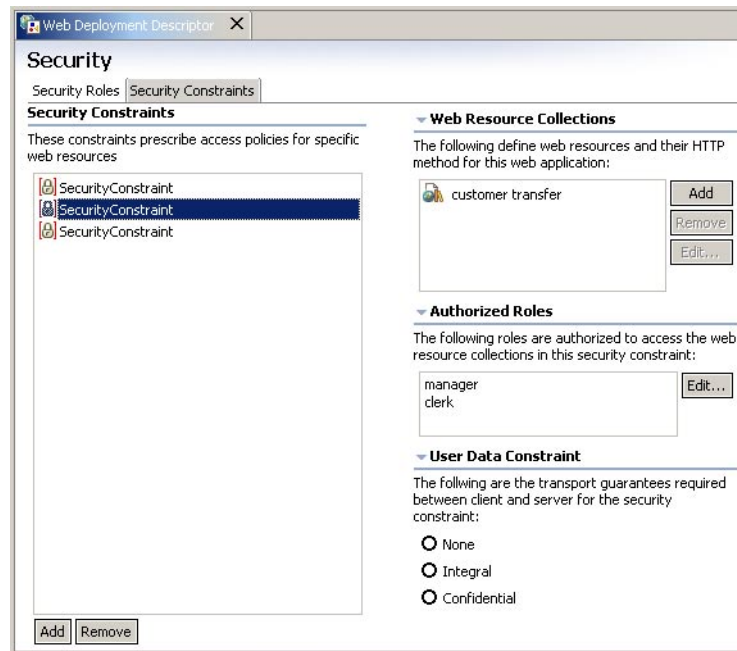
Figure 4-8   Security constraints editor in Studio
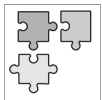
4.4 Security role reference

During the development phase of the application the actual role names for security constraints may not be known to the groups of developers. From the other hand the actual role names in a deployed runtime environement may 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.
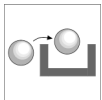
Security role references provide a level of indirection to isolate roles used during development and actual runtime roles. They link the names of the roles used oin the module to the corresponding name of the role in the encompassing application.

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. The Application Assembly Tool (AAT) and WebSphere Studio Application Developer can be used to both define the role-name and map them to the actual runtime roles in the environment with the role-link element.
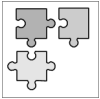
Security Role References with Application Assembly Tool

Example below provides instructions on how to define role references using the Application Assembly Tool.

1. Start the Application Assembly Tool, open the **itsobank.ear** archive.

2. On the right hand side expand the tree: **itsobank -> Web modules -> itsobankWeb -> Web components -> TransferServlet -> Security Role References.**

3. Right-click the **Security Reference** node, then select **New**.

4. A window pops up with the settings. A Name specifies the name of a security role reference used in the application code; type in **RoleRef**.

5. Link specifies the name of a security role defined in the encompassing application, in our example it is a application Web module; select **manager** here.

6. You can write a description for the entry if you need to, actually this might be a good place to put a description for future references.

7. Click **OK**.
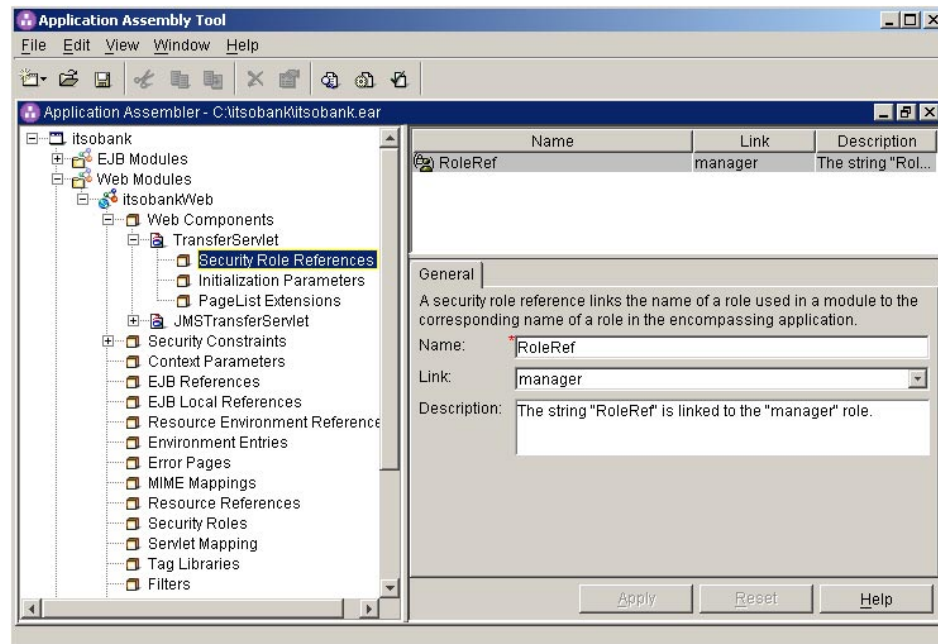
Figure 4-9   Defining Security role references in Applicaiton Assembly Tool

Security Role References with WebSphere Studio

The following steps will show, how to define Security Role references for the Web module in WebSphere Studio Application Developer.

1. Open to the **Web perspective** in Studio, select the **J2EE Navigator**

2. Expand **itsobankWeb** tree and double click **Web deployment descriptor**. This will open the editor with the **web.xml** file.

3. Switch to the **Servlets** tab.

4. Select **TransferServlet** in a Servlets and JSP section.

5. In the **Authorized Roles** section click **Edit**. This will open Select Authorized Roles dialog box.

6. Select the role that you want to add, for example: `manager` and `clerk`.

    This performs the reference a bit differently than the Application Assembly Tool; in this case, the name in the source code and the link to the application roles will automatically have the same reference name; in our example the reference to manager is manager, the reference to clerk is clerk. You can check the entry in the web.xml source by searching for the following tag: *<security-role-ref>.* This is not the case in Application Assembly tool, where actual reference name and link name can be different.

7. Save and close the web.xml file.
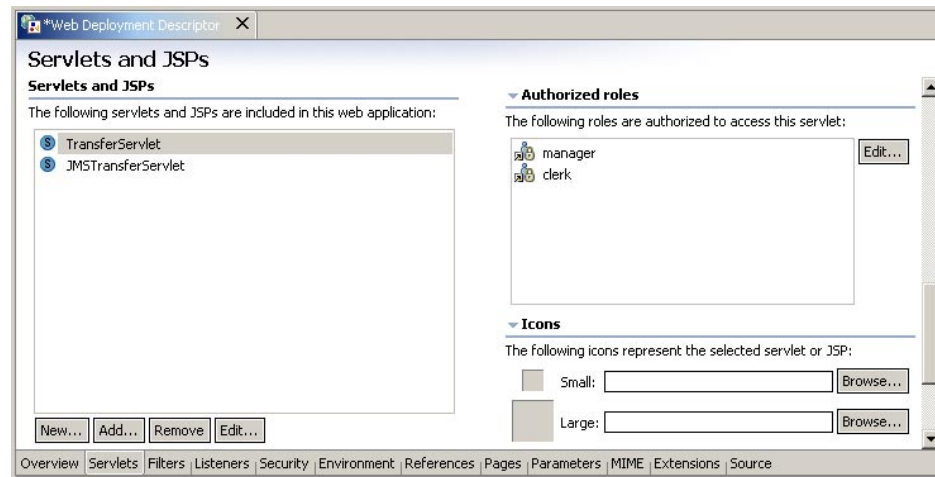
Figure 4-10   Defining security roles references in Studio

Note: Roles visibility within .ear file

In the notion of roles WebSphere 5 differs from WebSphere 4 in the subject of roles visibility. Each module can have independent logical roles. This means that if we define runtime roles on the application level they do tor roll down and are not visible in any application module. However roles defined for modules will roll up and be visible on the application level. This wil allow to create global references from the application down to the Web and EJB modules.

4.5 Login facilities

J2EE specification defines the following types of types of authentication methods.

► Basic authentication

► Digest authentication

► Form based authetication

► Client Certificate based authetication

For a brief description of each type of authentication see Chapter 4.2.1, "Configuring Web module security" on page 45.

For any types of authethnication methods to work, at least one security constraint should be defined for the requested Web resources and Global Security must be enabled for the application server.

For instructions of how to define security constraints for Web resources see Chapter 4.2.1, "Configuring Web module security" on page 45.

For instructions of how to enable Global Security on the server please refer to Section 10.2, "WebSphere Global Security" on page 235.

When developing WebSphere applications you can configure authentication methods using either WebSphere Studio or Webpshere Application Assembly Tool. This chapter presents basic scenarios of how to set up login Authetication methods for the ITSObank application.
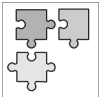
4.5.1 Form based login

One of the login challanges defined in J2EE specification is Form Based Login. It enables the application developer to customize login process and present an application specific form by making use of the Form Login Authentication Method.

Form Login works in the following manner:

1. Unauthenticated user request a resource protected by Form Login authentication type.

2. The application server redirects the request to the Login Form defined before in the Web deployment descriptor.

3. On the HTML login form user enters the user ID and password and submits the form.

4. The action triggered by form submission runs a special WebSphere Application servlet **j_security_check**. Web container after receiving a request for the **j_security_check** servlet dispatches the request into another WebSphere servlet that authenticates the user.

5. If the servlet authenticates the user succesfully, originaly requested resource is displayed.

If you select LTPA as the authentication mechanism under global security settings and use form login in any Web applications, you must also enable Single Sign-On (SSO). 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 at the Web browser. This information is needed to authorize protected resources when a form login is used.

Form Login Configuration

The following steps shows how to configure form based login using the Application Assembly Tool and Studio.

Using Application Assembly Tool

Detailed steps for the Application Assembly Tool:

1. After importing the itsobank.ear file into AAT expand the tree **itsobank->Web Modules** and right-click the **itsobankWeb** Web module.

2. In a pop-up menu select **Properties.**

3. Switch to the **Advanced** tab.

4. Check the box **Login Configuration**.

5. Click drop-down list **Authentication Method** and select **FORM**.

6. In the **Realm Name** field enter `ITSOBank`

7. in the **Login page** field enter `/login.html`

8. in the **Error page** field enter `/loginerror.html`

9. Click **OK** to close **Web Modules Properties** window.

Using WebSphere Studio

Detailed steps for WebSphere Studio Application Developer.

1. Expand the tree **itsobankWeb -> Web Content -> WEB-INF** and double click the file **web.xml**. A Web Deployment Descriptor should be opened in a deployment descriptor editor window.

2. Select **Pages** tab then modify the Login section:

   a. Write realm name, we have used `ITSOBank`.

   b. Click the drop-down list and select `FORM` as the Authentication method.

   c. In Login page click browse and select `login.html`.

   d. In Error page click browse and select `loginerror.html` (we have used the same page for login and error. You can define custom error.jsp page that will present actual error code and error message).

3. Click **File -> Save Deployment Descriptor.**

Setting Authentication Method for the application Web module will create *<login-config>* section in a Web deployment descriptor XML file. See example below.

Example 4-4   Login-config section of the Web deployment descriptor

```
<login-config>
    <auth-method>FORM</auth-method>
```

```
    <realm-name>Webbank realm</realm-name>
    <form-login-config>
        <form-login-page>/login.html</form-login-page>
        <form-error-page>/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. Servlet j_security_check used by WebSphere Application Server enforces only the name of the input fields that the developer should put on the custom Login Form.

These fields are:

▶ **j_username** should be the input field in which a user will write the user ID.

▶ **j_password** should be the input filed into which user will write the password.

The action required for the HTTP POST method is j_security_check. Simple HTML code for custom Login Form is givent in the Example 4-5 on page 60.

Example 4-5   Sample custom login form from the ITSOBank application

```
<!-- .............. -->
<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"> <input type="reset"
name="reset" value="Clear">
</td></tr></table>
</form>
<!-- .............. -->
```

Note: The *j_security_check* servlet will not work when global security is disabled, the application server will return a page not found error.

It is also true for the *ibm_security_logout* servlet, which you will see later.

4.5.2 Custom login

There are situations when the login facility, using the j_security_check servlet, provided by WebSphere does not fulfil all the requirements for the application. In these cases developers can extend the login facility and develop an extension to the existing code.

In the earlier versions of WebSphere Application Server, developers could use the Custom Login facility of WebSphere; which has been deprecated since version 4 of the WebSphere Application Server.

According to the new programming model, developers should use servlet filters to implement pre-login or post-login processes.

The following section will give a short introduction of servlet filters then it will provide a sample filter to do post-login processing for the application.

Using servlet filters to modify login process

Java Servlet API version 2.3 introduces a new object called filter that can transform a request or modify a response or header information. Filters can be chained together to act on the input and output of a specified resource or group of resources. They do not usually create a response. The main role of filters is to modify or adapt the response. Typical use of filters inculdes:

1. Logging information.
2. Transforming the content on the fly (image transformation, encryption, XML transformation, compression, and so on).
3. MIME type filters (funtionaly equivalent to the old-style servlet chaining).
4. Customized authetication of Web resources.
5. Caching information.

Filter can be configured to act upon a certain request. A difference between JSP/Servlet and filter processing is that filter can be mapped and work across the subset (or all) of the URLs served by application.

Filters lifecycle is very similar to servlets'. Configuration of all filetrs in a given Servlet conatainer is kept in the FilterConfig object. Each filter can access this object in order to get the initialization parameters, a reference to the ServletContext and to load the information necessary to perform filter processing (for example data needed for filtering functions).

Each object that will be used as a filter should implement the Filter interface. This interface defines three methods:

▶ `public void init(FilterConfig filterConfig)`

A method called by Web container to initialize FilterConfig object for the filer and to ensure that the filter is being instantiated.

▶ `public void doFilter(final ServletRequest request, final ServletResponse response, FilterChain chain)`

A method called every time when the request/response pair is passed through the filters.

▶ `public void destroy()`

A method called by the container to clear the instatnce of the filter. This method can be used to clean up all the resources that were kept for filter processing tasks.

When planning a scenario for filters you need to take into account the way how filters work.

Actions performed by servlet filter maybe executed before and/or after the actual servlet, or JavaServer Page. When thinking about login process, servlet filter may perform some pre-login functions before sending the request to the authentication servlet. From the other hand it may take the result of the authentication servlet and perform additional checking for example in external databases in order to send customized response to the client's browser.

As mentioned in Chapter 4.5.1, "Form based login" on page 58 WebSphere Application Server uses the special *j_security_check* servlet to perform authetication when form based authentication is selected for the Web application.

This section will present a sample filter that is assigned to *j_security_check* servlet to perform additional LDAP lookup and in order to retrieve attributes for the user who logged in.

This scenario assumes:

1. WebSphere Application Server is configured with security enabled. The type of user registry used for that scenario does not make any difference to this implementation, any user registry can be used with the sample, however when other user registry is used than LDAP, make sure that each user from the user registry of your choice exist in the LDAP directory for the additional lookup.

2. Servlet filter will communicate with LDAP server in order to get certain user attributes. User description in LDAP server contains employee type attribute that will be checked by servlet filter. The value of the *employeeType* will be stored as a session attribute and used by other Web components. Specify the employeetype attribute for the application users in the LDAP directory.

3. It assumes that LDAP is configured to accept anonymous access.

4. The example implemented in this book have the users defined in both local operating system user registry and in LDAP server. Users are registered under the o=itso suffix.

Developing the LDAP query code

The example filter included in the application code contains the following java files.

▶ **LDAPAttr.java**, is utilized to connect to the LDAP server and retrieve user attributes. Parameters that are passed to the class are set up in the filter configuration section in Web deployment descriptor and are passed to the construcor of the class while it is created. The class implements the simpliest way to read user attributes from LDAP server.

▶ **PostLoginFilter.java** is the actual filter code that uses the LDAPAttr to access certain attributes in the LDAP directory. This class implements the *doFilter(request, response, filterchain)* method. In this method the actual filter action is performed, it is also the method that upstream filter calls in order to pass the processing to the next filter in a chain. FilterChain object provides the information about the next filter to call.

The sample doFilter method does the following:

a. Checks if Web container successfuly initialized the filter. Web container calls init method of the filter to instantiate it.

b. Reads the init parameters of the filter that have been provided in the **<filter>** section of the Web deployment descriptor.

c. Gets the user name from the HTTP request object and creates the session attribute in the HttpSession object. This attribute is initally set to UNDEFINED.

d. The filter lets the j_security_check to perform the actions by calling doFilter method.

e. After returning from j_security_check, the PostLoginFilter performs an LDAP search for the user name that was provided in the HTTP request to the login form.

f. After succesfull search the user in LDAP directory, the employeeType attribute is read for the user and session object is updated with the result.

Any other Web component during the session can access the object to get the attribute.

Part of the Java code for **PostLoginFilter** represents the steps described above.

Example 4-6   PostLoginFilter sample code

```
public void doFilter(ServletRequest request,ServletResponse response,
```

```
        FilterChain chain)
//.....
// Create attribute in HttpSession object and set its value to "UNDEFINED"
HttpSession session = ((HttpServletRequest) request).getSession();
session.setAttribute("userTYPE", "UNDEFINED");

//let the j_security_check to do it's work
chain.doFilter(request, response);

//.....
//perform LDAP retrieval
userLDAPAttributes = new LDAPAttr(cfg_server, cfg_port, cfg_basedn);
userLDAPAttributes.SetUser(userName, cfg_attr);

if (userLDAPAttributes.RetrieveLDAPAttributes() == 0) {
    user_attr = userLDAPAttributes.GetUserAttr();
    //Update session object attribute
    session.setAttribute("userTYPE", user_attr);
    System.out.println("Attrbute in the session object was set to: "
        + (session.getAttribute("userTYPE")).toString());
else {
    filterConfig.getServletContext().log(
    "Problems retrieving attributes for " + userName);
}
//.....
```

In the filter source code, extensive comments have been provided in order to clearly explain the filter behavior. Please refer to the filter code in the application for more information.

Filter configuration

We have used WebSphere Studio Application Developer in order to install and configure the filter in the Web deployment descriptor.

1. Open the Web deployment descriptor in WebSphere Studio.

2. Select the **Filters** tab and click **Add** under the Filters section. The Filter Type Selection windows should be opened.

3. From the Filter Type Selection window select the **PostLogin** filter class and click **OK**. PostLogin filter should now appear. Details section should contain the full name for the filter class.

4. In the **URL Mappings** section click **ADD** then write */j_security_check*.

5. In the initialization section enter the following parameters:

   ServerName: `ldap://dirsrv01`, replace with your directory server's URL.

   ServerPort: 389

BaseDN: `o=itso` (This is the context root where the filter will look for the user)

UserAttr: `employeetype` (this is the attribute that the filter will check)

Finally your Filters tab should look like the picture below.
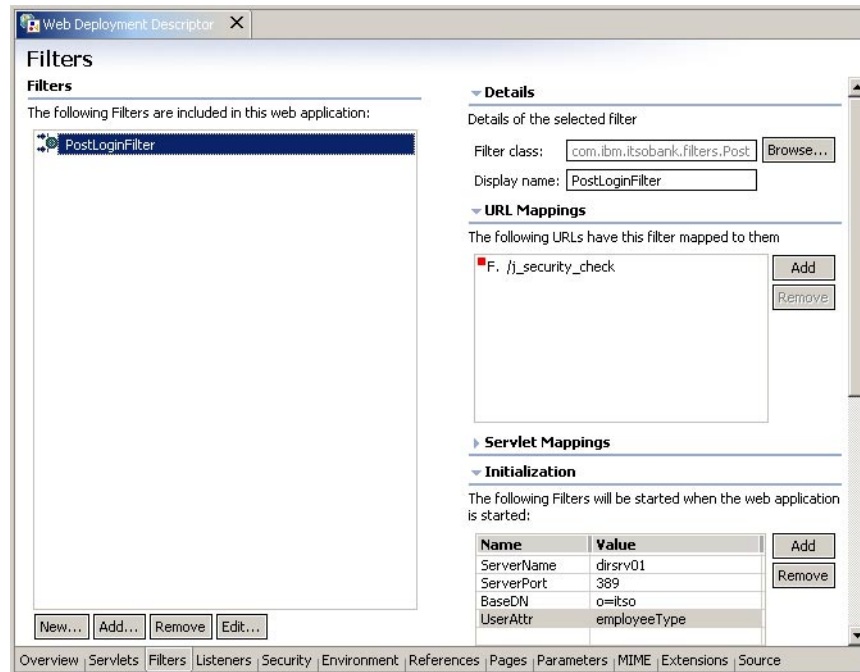
Figure 4-11   Filter configuration window in Studio

Saving the file should update filter configuration section in the Web depolyment descriptor, web.xml, file.

Testing the application

First of all, modify the LDAP user entries to store the employee type value. If you are using SecureWay Directory Server start the Directory Management Tool (DMT). Rebind the connection using authentication, and login as cn=root. Find the **manager01** user entry in the directory tree and double-click it. Change the emplyee type to: regular employee, then click **OK**.

Figure 4-12   LDAP user entry

Do the same modification for the other users, according to the following table.

Table 4-1   Employee type settings for users

Username	Employee type
accountant01	partner
clerk01	contractor
consultant01	regular employee

As a next step you have to make sure that the application is using the right LDAP settings for the filter. Actually, you will have to change them in the .ear file and redeploy the ITSOBank application in WebSphere. In order to avoid all the security mappings you can simply export the application from WebSphere and do the modifications on the exported .ear file.

Open the ITSOBank application in the Application Assembly Tool and select the **itsobank -> Web modules -> itsobankWeb -> Filters -> PostLoginFilter -> Initialization Parameters.** Modify the filter initialization settings: BaseDN, ServerName, ServerPort, UserAttr to reflect your runtime configuration, then click **Apply** then save the .ear file. Use this latest version of the .ear file and deploy it in WebSphere.

To test the custom login implemented in this section, open the sample ITSOBank application, that comes with the book, in your browser at http://<your_server>/itsobank.

Select the link on the main page which says: **Modified Customer Transfer**. When you have security enabled the application returns the login page first. Once you have logged in with the right user, you will see the employeeType value at the bottom of the customertransfer.jsp page.

4.5.3 Form based logout

One of the IBM's extensions to J2EE specification is the form based logout. 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 with caling a POST action on the *ibm_security_logout* servlet. This form must exist within the same Web application to which the user gets redirected after logging out.

Example 4-7   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.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 the most intelligent solution to finish a session with an application.

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

Note: The previously introduced logout only works together with form-based login. When the application is configured to use Basic Authentication (BA) the credentials are stored in the client's browser and the browser will send 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.

4.6 Additional security guidelines

This section provides some useful information or gives some thoughts on security considerations and issues. These short articles, paragraphs cover the following topics:

► Security constraints for the Web module

► Struts security

► Page expiration
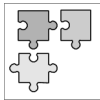
Security constraints for the Web module

There are multiple approaches to set authorization rules for resources. In this case we will investigate the possibilities for Web resource protection using Web security constraints in the Web module.

The first approach would be to map resources to roles, and define what resources a role can access. If you design your Web security constraints to use one constraint for every role and you have the same resource accessible by multiple roles, the security will not work correctly because of bad design.

According to the Java Servlet 2.3 specification, exact or logest path match is used to get the required roles for a given URL. Let's consider the following situation:

We have two roles defined for the application: A and B.

We have the following Web resources in the Web module:

– /helloworld/helloEurope.html

– /helloworld/helloAfrica.html

We have the following Web security constraints in the Web module:

– Constraint 'X': role: 'A' can access the resource(s): /dir/*

– Constraint 'Y': role: 'B' can access the resource(s): /dir/helloEurope.html

If user in role 'A' accesses /helloworld/helloEurope.html gets an authorization error saying that the user is not in role of 'B', however we set that role 'A' should be able to access everything under /helloworld.

User in role 'A' can still access the /helloworld/helloAfrica.html resource, and user in role 'B' can access /helloworld/helloEurope.html

As you see Constraint 'Y' overrules Constraint 'X' in the previous situation.

The other approach would be to map roles to resources, and define the authorized roles for each resource. Obviously this is not a feasible solution since an application probably has more resources than we want to set up one-by-one.

A solution can be, the same we applied to the ITSOBank sample application, is to reuse the use cases for the application and follow them to define security constraints. In this approach each constraint covers a use case, the roles are the identified actors for the use case and the resources are the resources involved in the use case.

For example, the ITSOBank sample application has a use case: customer transfer. The actors that can use this use case: manager and clerk. The resources are: transfer/customertransfer.html, servlet/Transfer, transfer/transferresults.jsp. The listed elements can define the appropriate resource collection for the right group of roles. Of course it is only one approach and it might not be the best in every case.

The purpose of this section is to point out the problem with the first two approaches and make you think about this issue.

You can also protect your resources based on URL patterns using a security reverse proxy in your infrastructure, for example: Tivoli Access Manager's WebSEAL.
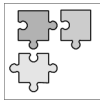
Struts security

Struts is a very powerful framework to implement the Model-View-Controler design pattern for a Web application. The framework at this stage does not provide security functions, it leaves the issue to the J2EE Web module to handle security. Struts does not carry any security problems, but there are certain considerations or thoughts you have to keep in mind.

The reason why the security issue rises is because Struts is a single access point under the cover, for multiple application functions. One single servlet handles all actions implementing the command pattern.

Struts supports multiple mechanisms to map URIs for different actions; there are two common approaches to define the URIs that will be processed by the controller servlet - prefix matching and extension matching.

▶ Prefix matching means that you want all URI that start with a particular value to be passed to this servlet, for example:
`http://www.xyz.com/myapp/execute/myaction`.

▶ Extension mapping, on the other hand, matches request URIs to the action servlet based on the fact that the URI ends with a period followed by a defined set of characters. For example, the JSP processing servlet is mapped to the *.jsp pattern so that it is called to process every JSP page that is requested. In case of struts we can map actions to the .do extension, that implies, "to do something"; the URL looks like the following:
`http://www.xyz.com/myapp/myaction.do`

In any cases the actions are differentiated based on URIs, although only one servlet performs the action. You have to make sure that you protect all the required URIs for your application for all the actions with Struts. Because you will only have only one servlet mapping, that does not mean you only have to set up one constraint. The only difficulty with Struts and with these approaches is that you have to design and administer your security constratints for your Web application very carefully; just like in any other case of security design. It is easy not to secure or set wrong access control to a sensitive resource and leave a security hole in your application.

You can also protect your resources based on URL patterns using a security reverse proxy in your infrastructure, for example: Tivoli Access Manager's WebSEAL.

Page expiration

This short paragraph is only a useful tip that you might want to keep in mind. The pages that are served from application servers are cached at least on the client side, no matter if the resource was secured or not.

It will be a problem in a live system, where sensitive information can be found in the cache.

It is definitely a problem during development, when security settings were changed for the Web module but the Web browser is still showing the pages from the cache without going out to the server and get the latest content, therefore no authentication or authorization check is performed. In this case you can simply set your browser to not to cache pages and always send a request.

As a universal solution for the problem, you can set expiration time, for example: "0" (zero) that means no caching, for all the secured pages on your application server. The following HTML code at the beginning of your page takes care of page expiration.

```
<META HTTP-EQUIV="expires" CONTENT="0">
```

The following code tells the Web browser not to cache the page:

```
<META HTTP-EQUIV="cache-control" CONTENT="no-cache">
```

4.7 Where to find more information

For more information about J2EE, servlets and JSPs, refer to Sun's Java Web site at `http://java.sun.com`.

► The J2EE 1.3 specification is available at:
   `http://java.sun.com/j2ee/docs.html`

► The Servlet 2.3 specification and the JSP 1.2 specification is also available at the previous URL.

5

Securing EJBs

This chapter focuses on security aspects of the EJB container, including:

► An overview of EJB security.

► How to protect EJB methods by assigning method permissions.

► How to assign and use security role references for EJBs.

► How to allow EJBs to make calls using a delegated identity.

6573chSecEJB.fm Draft Document for Review November 6, 2002 2:37 pm

5.1 Securing EJBs

EJBs, or *Enterprise Java Beans* are J2EE components which implement the business logic of an application. They typically have access to sensitive data, and it is very important to understand how security is applied to these resources.

There are three types of EJBs:

1. Session Beans, which represent clients inside the J2EE server. Clients call session bean methods to access an application.

2. 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.

3. Message-Driven Beans, which allow J2EE applications to process messages asynchronously. Message-driven beans' methods are invoked by the application server runtime as part of message queue processing.

> **Important:** Since queued messages generally do not have any authentication information associated with them, authentication information is unavailable to message-driven beans' 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 which need to be aware of the role or identity or the caller can programmatically call the J2EE API methods `isCallerInRole` and `getCallerPrincipal` to determine a caller's role and principal, respectively. When using `isCallerInRole`, *security role references* are used, which are later mapped to security roles.

> **Note:** If Websphere security is not enabled, or if the EJB is not a protected resource, isCallerInRole will return false and getCallerPrincipal will return UNKNOWN.

> **Note:** See below for details on how security role references are administered for EJBs. Programmatic security is covered in detail in Chapter 8, "Programmatic security" on page 179.

► Session, entity, and message-driven bean methods can be delegated to execute under the identity of either the caller (default), the EJB server, or a specific security role. This is refered to as the *Delegation Policy* or *Run-As Mode Mapping.*

In the next sections, each of these methods of applying security to EJBs will be discussed in detail.
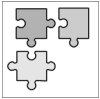
5.2 Defining J2EE roles for EJB modules

The method for defining security roles for EJBs and Web Components in the Application Assembly Tool is the same. For example, to add a role named *manager* to the EJB component, do the following:

1. Open the .ear file of the application, in our example: `itsobank.ear`.

2. Open the **EJB Modules** folder for your application, and select **Security Roles .**

3. If no security roles have previously been defined for EJBs, the box on the right will be empty. Right-click in the space under Name, and you will see the pop-up menu shown in Figure 5-1. Select **New** to create a new security role**.**
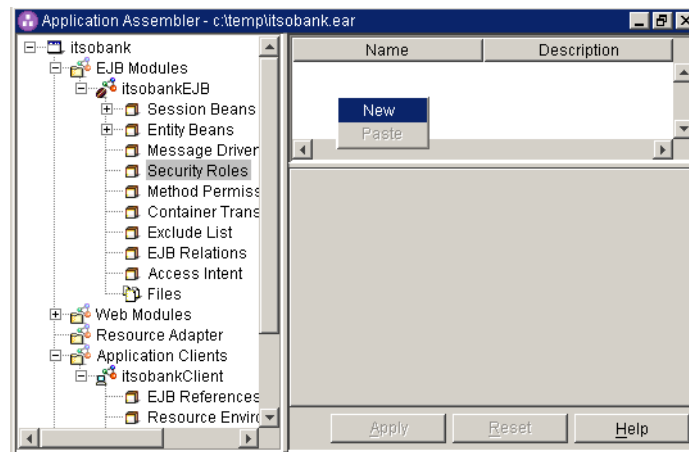
Figure 5-1   Creating a new Security Role

4. In the New Security Role dialog, see Figure 5-2, enter the name of the role, Manager, and (optionally) a description of the role.
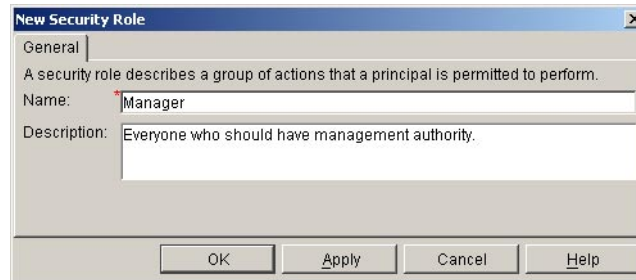
Figure 5-2   Application Assembly Tool - New Security Role Dialog

5. Click **Apply** if you wish to add more security roles to the EJB component, or click **OK** to close the New Security Role Dialog.
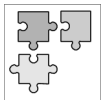
5.3 Assigning EJB Method Permissions

Session and entity bean methods can be secured by preventing access to all but members of the security roles which need to access those methods. These method permissions can be applied using either the Application Assembly Tool or the Websphere Studio.

The method permissions are included in the application deployment descriptor file ejb-jar.xml. For example, the following example shows the XML elements which would allow members of the manager role to call all methods in the BranchAccount EJB, all Local Home methods in the CustomerAccount EJB, as well as the create() and remove() methods in the Consultation EJB.

Example 5-1   Method Permissions in the ejb-jar.xml file

```
<method-permission id="MethodPermission_1">
    <description>manager method permissions:+:</description>
    <role-name>manager</role-name>
    <method id="MethodElement_1">
        <ejb-name>Consultation</ejb-name>
        <method-intf>Home</method-intf>
        <method-name>create</method-name>
        <method-params></method-params>
    </method>
    <method id="MethodElement_2">
        <ejb-name>BranchAccount</ejb-name>
        <method-name>*</method-name>
    </method>
    <method id="MethodElement_3">
        <ejb-name>Consultation</ejb-name>
```

```
        <method-intf>Home</method-intf>
        <method-name>remove</method-name>
        <method-params>
            <method-param>javax.ejb.Handle</method-param>
        </method-params>
    </method>
    <method id="MethodElement_4">
        <ejb-name>CustomerAccount</ejb-name>
        <method-intf>LocalHome</method-intf>
        <method-name>*</method-name>
    </method>
</method-permission>
```
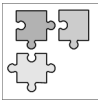
Assigning Method Permissions in the Application Assembly Tool

To set up these method permissions using the Application Assembly Tool, do the following:

1. Open the **EJB Modules** folder for your application, and select **Method Permissions.**

2. If no method permissions have previously been defined for EJBs, the box on the right will be empty. Right-click in the space under Name, and you will see a pop-up menu. Select **New** to create a new set of method permissions.

3. In the **New Method Permission** dialog, you may enter a **Method permission name** and **Description**, although these are optional.

4. To add methods, click the first **Add...** button and you will see the **Add Methods** dialog. By opening the folders, you can see all methods in all the application's EJBs. To select multiple methods, hold down the control key as you select methods. All EJB methods of a given type, or all methods for a given EJB can be selected by selecting the parent folders as shown in Figure 5-3.

5. Click **OK** when done.
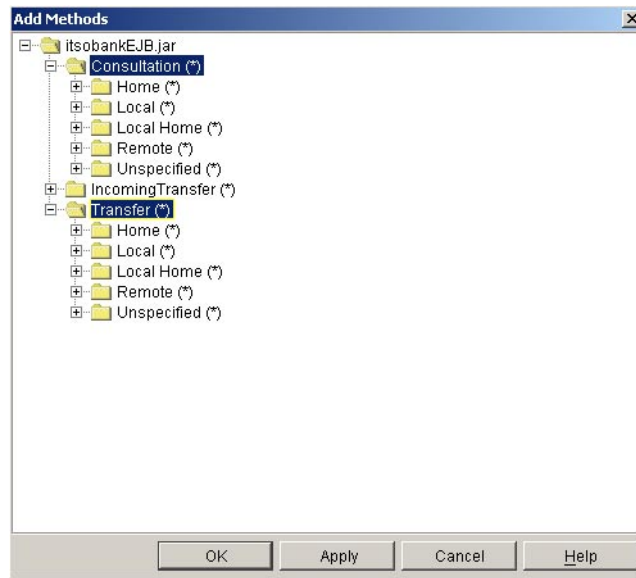
Figure 5-3   Add Methods Dialog

6. To assign the selected methods to a security role, click the **Add...** button next to the **Role Name** list. You will see a list of all security roles that have been defined in the EJB module. For information about adding security roles, see Section , "Defining Security Roles in the Application Assembly Tool" on page 28.

7. Select the **manager** security role and click **OK**. Now, the New Method Permissions dialog will appear as in Figure 5-4, and the resulting deployment descriptor will contain the XML code shown in Example 5-1 on page 76.
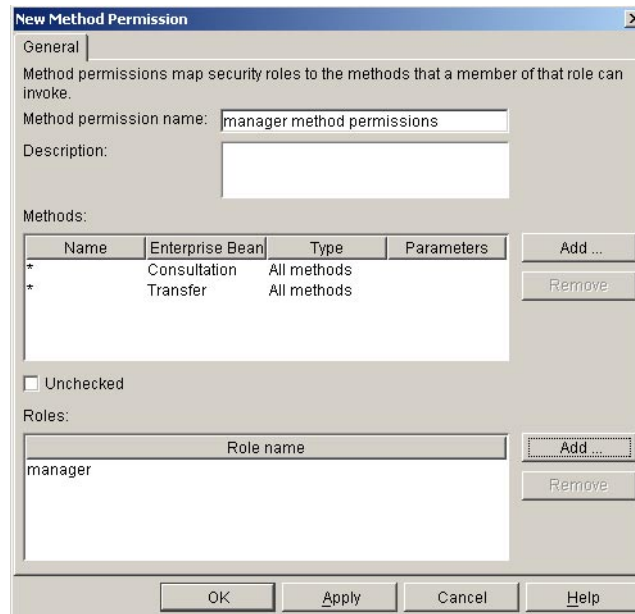
Figure 5-4   New Methods Permission Dialog

Assigning Method Permissions in WebSphere Studio

To set up these method permissions using the Application Assembly Tool, do the following:

1. From the Resource Perspective, navigate to the EJB deployment descriptor file, **ebj-jar.xml,** and double-click this file.

2. Click the **Assembly Descriptor** tab to see the method permissions.

3. If no security roles have been defined for EJBs, click the **Add...** button below the **Security Roles** box, to see the Add Security Role dialog. Enter a **Name** and (optionally) a **Description** and then click **Finish**. Repeat to add all necessary security roles to the EJB module.

4. click the **Add...** button below the Method Permissions box to see the Add Method Permissions dialog. Select one or more security roles, as shown in Figure 5-5.
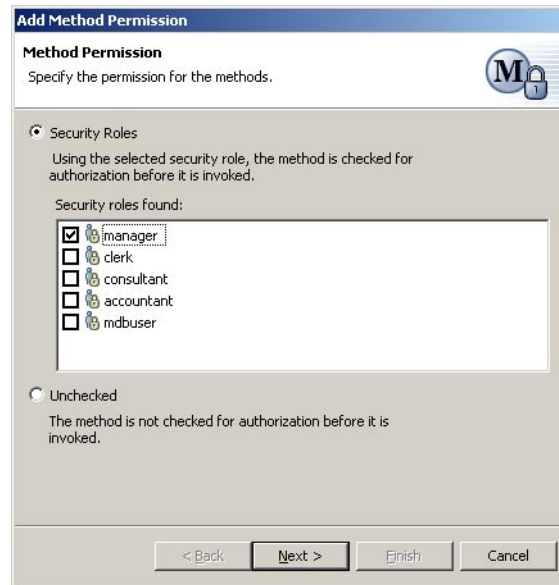
Figure 5-5   Choosing Security Roles

5. Click **Next...** to see the list of EJBs. Select one or more EJBs from the list.

6. Click **Next...** to see the list of methods. Select one or more methods, using the wildcards (*) if desired to include all methods of a given type or all methods for a given EJB.

7. Click **Finish** when done.
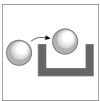
Assigning Roles to Unprotected Methods

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 unchecked.

These unprotected methods can have one of three permissions applied:

1. *Unckeck*. This is the default, and indicates that unprotected methods should be left unprotected. Anyone can call these methods.

2. *Exclude.* Unprotected methods are unavailable to all callers.

3. *Role.* Unprotected methods are available only to members of a specific security role.

Figure 5-6   Assigning roles to unprotected EJB methods

Note: This behavior is different than in previous WebSphere versions. In Websphere Application Server Version 4, the default was to grant access to all EJB methods when no methods were explicitly protected, and to deny access to all EJB methods (by default) when at least one EJB method was protected.

Most importantly, the default in Version 5 is for methods that are not explicitly unprotected to be *unchecked*.
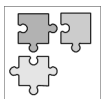
5.4 Security role references

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 in the application code.

When an EJB uses the IsCallerInRole(Java.lang.String roleName) J2EE API method to determine whether or not 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. For example, the following Java code shows how a security role referenced might be used.

Example 5-2   Security Role Reference Example

```
public String isInRole() {
    if (mySessionCtx.isCallerInRole("RoleRef")) {
        return "Caller is a member of the referenced role";
    } else {
        return "Caller is NOT a member of the referenced role";
    }
}
```

The following XML code shows how the security role reference *RoleRef* would be linked to the security role *manager*.

Example 5-3   Security Role Reference in `ejb-jar.xml`

```
<security-role-ref>
    <description>security role reference RoleRef is linked to security role
manager</description>
    <role-name>RoleRef</role-name>
    <role-link>manager</role-link>
</security-role-ref>
```

For a security role reference to work, 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.
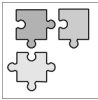
Security Role References in the Application Assembly Tool

To link the *RoleRef* security role reference to the *manager* security role using the Application Assembly Tool, do the following:

1. Open the **EJB Modules** folder for your application, and navigate to the **Security Role References** view for the EJB containing the method which calls `isCallerInRole`.

2. If no security role references have previously been defined for EJBs, the box on the right will be empty. Right-click in the space under Name, and you will see a pop-up menu. Select **New** to create a new security role reference.

3. In the New Security Role Reference dialog (see Figure 5-7), enter the reference's **Name**. This is the string that is passed to isCallerInRole() in the Java code.
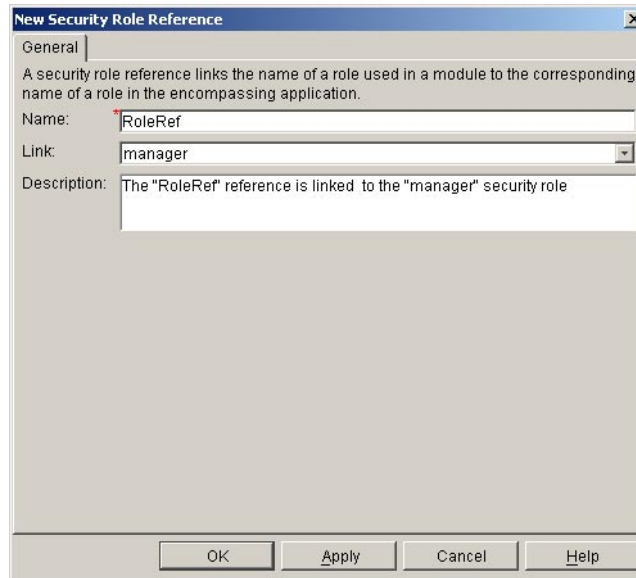
Figure 5-7   New Security Role Reference Dialog (Application Assembly Tool)

4. Select the desired security role from the Link pull-down menu. Only security roles which have previously been defined in the EJB module are shown in this menu.

5. Optionally, enter a **Description** for this security role reference.

6. Click **OK** to apply the changes and close the window.

Security Role References in the Websphere Studio

To link the RoleRef security role reference to the manager security role using the Websphere Studio, do the following:

1. From the Resource Perspective, navigate to the EJB deployment descriptor file, **ebj-jar.xml,** and double-click this file.

2. Click the **References** tab.

3. Select the bean containing the method which calls isCallerInRole() and click **Add...**

4. In the Add Reference dialog, select **Security Role Reference** and click **Next**.

5. In the Add Security Role Reference dialog, see Figure 5-8, enter the reference's **Name**. This is the string that is passed to isCallerInRole() in the Java code.
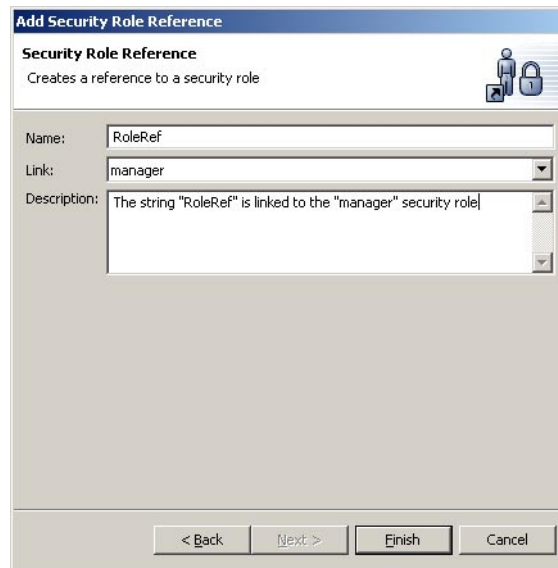
Figure 5-8   Add Security Role Reference Dialog in Websphere Studio

6. Select the desired security role from the Link pull-down menu. Only security roles which have previously been defined in the EJB module are shown in this menu

7. Optionally, enter a **Description** for this security role reference.

8. Click **OK** to apply the changes and close the window.
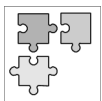
5.5 Delegation policy

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 getCallerPrincipal(). Occasionally, however, it is desireable for one EJB to call another with a previously defined identity, for instance one that is a member of a specifc role.

One example is in the case of a message-driven bean's onMessage() method which calls a protected method in an entity bean. Since 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 noticable 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.

5.5.1 Bean Level Delegation

The EJB 2.0 Specification defines delegation at the EJB bean 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 will be called using the identity of this mapped user.
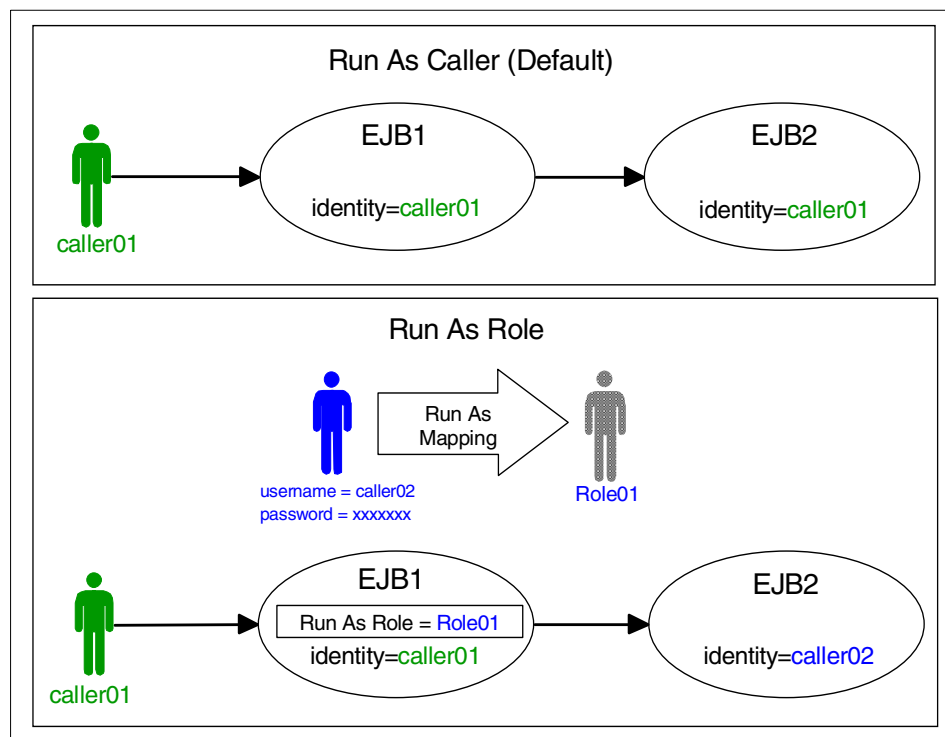
Figure 5-9   Run as Caller vs. Run as Role

Figure 5-9 demonstrates 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*.

The following example shows the XML code in the ejb-jar.xml deployment descriptor file for the default mode (run as caller):

Example 5-4   ejb-jar.xml code for non-delegated EJB

```
<security-identity>
    <description>This bean requires no delegation</description>
    <use-caller-identity />
</security-identity
```

The next example shows the XML code in the ebj-jar.xml file for a bean which has been delegated to run a a member of the *mdbuser* security role:

Example 5-5   ebj-jar.xml code for EJB that is delegated to run as role: mdbuser

```
<security-identity>
    <description>This message-driven bean calls protected methods in other
beans.</description>
    <run-as>
        <description>The methods of this bean run as a member of the mdbuser
role</description>
        <role-name>mdbuser</role-name>
    </run-as>
</security-identity>
```

Assigning Bean-level Run-as Delegation Policies in Application Assembly Tool

Bean-level delegation policies can be assigned using either the Application Assembly Tool, or the Websphere Studio. To assign a Run-as Role to an EJB using the Application Assembly Tool, do the following:

1. Open the **EJB Modules** folder for your application, and navigate to the particular EJB to which you want to assign a delegation policy**.**
2. Click the **Security** tab to see the Run-as mode settings.
3. The default **Run-As Mode** is **UseCallerID**. Choose **UseSpecificID** to assign a Run-as Role.

4. The **Role Name** option menu will contain the list of defined security roles for the EJB module. Select the desired role.

5. Enter an optional description for the Run-as Role.

6. Click **Apply** to save the delegation policy.

Figure 5-10   Assigning bean-level Run-as Role in the Application Assembly Tool

Assigning Bean-level Run-as Delegation Policies in WebSphere Studio

To assign a bean-level Run-as role to an EJB using the Websphere Studio, do the following:

1. From the Resource Perspective, navigate to the EJB deployment descriptor file, **ebj-jar.xml,** and double-click this file.

2. Click the **Access** tab.

3. In the Security Identity (Bean Level) box, select the EJB to which you want to assign the delegation policy, and click **Add...**

4. Select **Use identity assigned to a specific role (below)**

5. In the Role name box, select the desired role from the option list. This list will contain all security roles which have been defined in the EJB module.

6. Enter an optional **Role Description**.

Chapter 5. Securing EJBs   **87**

7. Enter an optional **Security identity description**.

Figure 5-11   Adding bean-level run-as role in Websphere Studio

8. Click **Next.**

9. In the Enterprise Bean Selection dialog, select one or more beans that should use this delegation policy, and click **Finish.**

Note: In the Security Identity (Bean Level) box, a bean's security identity is listed as server identity when the bean is set to Run-as caller (the default). When a bean is assigned a run-as role, its security identity is listed as caller identity, and the assigned role appears in the User Specified Identity box at the right.

Also, the **Remove...** button does not work. To remove a bean-level run-as role, simply add a run-as caller policy to the EJB.
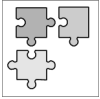
5.5.2 Method Level Delegation

In addition to the bean-level delegation policy defined by the EJB 2.0 Specification and described above, the Websphere Application Server provides the capability to do method-level EJB delegation. 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.

In addition, method-level delegation provides an additional delegation option: *run as Server*. This option indicates that the method should 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.xmi file. The following example shows the XML code for an onMessage() method which is delegated to run as the application server.

Example 5-6   Method-level run as server

```
<runAsSettings xmi:id="SecurityIdentity_3" description="Run this method using
the identity of the server">
    <methodElements xmi:id="MethodElement_3" name="onMessage"
parms="javax.jms.Message" type="Unspecified">
        <enterpriseBean xmi:type="ejb:MessageDriven"
href="META-INF/ejb-jar.xml#MessageDriven_1"/>
    </methodElements>
    <runAsMode xmi:type="ejbext:UseSystemIdentity"
xmi:id="UseSystemIdentity_2"/>
</runAsSettings>
```

The following example shows the XML code for an onMessage() method which is delegated to run as a member of the *mdbuser* security role:

Example 5-7   Method-level run as role

```
<runAsSettings xmi:id="SecurityIdentity_4" description="This message-driven
bean calls protected methods in other beans.">
    <methodElements xmi:id="MethodElement_4" name="onMessage"
parms="javax.jms.Message" type="Unspecified">
        <enterpriseBean xmi:type="ejb:MessageDriven"
href="META-INF/ejb-jar.xml#MessageDriven_1"/>
    </methodElements>
    <runAsMode xmi:type="ejbext:RunAsSpecifiedIdentity"
xmi:id="RunAsSpecifiedIdentity_1">
        <runAsSpecifiedIdentity xmi:id="Identity_1" roleName="mdbuser"
description="The methods of this bean run as a member of the mdbuser role."/>
    </runAsMode>
```

```
</runAsSettings>
```

Assigning Method-level Run-as Delegation Policies in Application Assembly Tool

Delegation policies as the EJB method level can be assigned using either the Application Assembly Tool or the Websphere Studio Application Developer.

To assign a run-as policy to an EJB method using the Application Assembly Tool, do the following:

1. Open the **EJB Modules** folder for your application, and navigate to the **Method Extensions** view for the EJB containing the method which you want to delegate**.**

2. Select the method(s) to which you want to assign the delegation policy. Wildcards (*) can be used to select all methods of a given type.

3. Click the **Advanced** tab and select the Security Identity checkbox to enable the Run-as mode selections.

4. Enter an optional **Description**

5. Select one of the following **Run-as** mode options:

   g. Use identity of caller (this is the default)

   h. Use identity of server

   i. Use identity assigned to specific role.

6. If assigning a run-as role, select the desired role from the **Role Name** selection list. This list will contain all roles which have been defined in the EJB module. The role description is optional
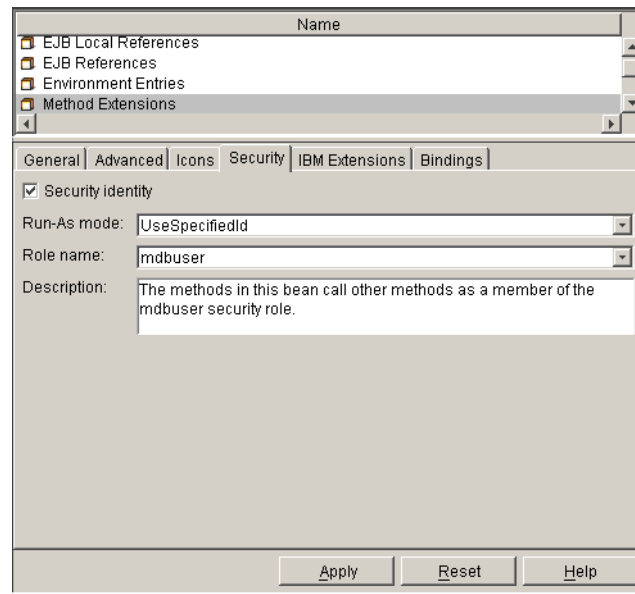
7. Click **Apply** to keep the policy settings.

Figure 5-12   Method-level run-as role in the Application Assembly Tool

Assigning Method-level Run-as Delegation Policies in WebSphere Studio

To assign a run-as policy to an EJB method using the Websphere Studio Application Developer, do the following:

1. From the Resource Perspective, navigate to the EJB deployment descriptor file, **ebj-jar.xml,** and double-click this file.

2. Click the **Access** tab.

3. In the Security Identity (Method Level) box, select the EJB to which you want to assign the delegation policy, and click **Add...**

4. In the **Add Security Identity** dialog, select the desired **Run-as** mode.

5. If using Run-as role, select the **Role Name** from the list of security roles previously defined in the EJB module. The description of the role mapping is optional.

6. Enter an optional description for the delegation policy.

Figure 5-13   Method-level run-as role policy in Websphere Studio

7. Click **Next.**

8. In the Enterprise Bean Selection dialog, select the EJBs containing the methods to which you want to assign this delegation policy, then click **Next.**

9. In the Method Elements dialog, select the EJB methods to which this delegation policy should be assigned.
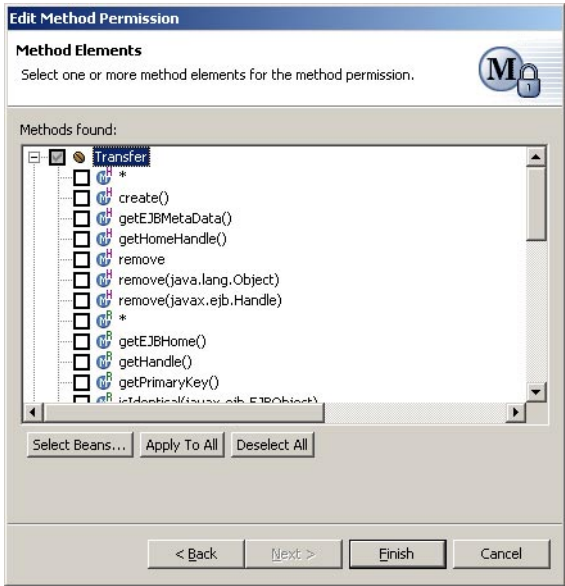
Figure 5-14   Selecting methods for the methods permission in WebSphere Studio

10.Click **Finish** when done.

5.6 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 very different from, but easily confused with, security role mapping. The following table compares these two concepts:

Run-as Mapping vs. Security Role Mapping

Run-as Mapping	Security Role Mapping
Run-as mapping is used to determine the principal from the user registry that will be 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 will be 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.

Run-as Mapping	Security Role Mapping
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 runtime.
Run-as mapping is performed using the Websphere Administrative Console only.	Security role mapping is performed using the Application Assembly Tool, the Websphere Studio, or the Websphere Administrative Console.
Cannot be modified after application installation	Can be modified after applicaiton installation using the Websphere Administrative Console.

When installing an application which defines either a bean-level or method-level run-as role delegation policy, one of the steps will be to map the run-as role(s) to a real user, as in Figure 5-15.

1. Select the **Role** that you wish to map.

2. Enter a valid **username** 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.

4. Once all run-as roles have been mapped to real users, click **Next** to continue installation.

Figure 5-15   Run-as role mapping

If one or more method-level delegation policies specify run-as system, one of the installation steps will be to verify this policy. The dialog appears as in Figure 5-16, and for each method that specifies 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 5-16   Verifying Run-as System

To override the run-as system mapping and assign a run-as role, do the following:

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.

3. Enter a valid **username** 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 the run-as role policy.

5. Click **Next** to continue with installation.

5.7 Where to find more information

For more information about J2EE, servlets and JSPs, refer to Sun's Java Web site at `http://java.sun.com`.

► The J2EE 1.3 specification is available at:
`http://java.sun.com/j2ee/docs.html`

► The EJB 2.0 specification is also available at the previous URL.

6

Securing Java clients

This chapter discusses how a variety of Java clients may be configured to access a secured server-based application. A Java client, in this context, is one which acts as an EJB client. The Java client may be operating on the same machine or a different machine to the EJB container and CORBA is used as the marshalling mechanism between client and server.

Relevant aspects of the OMG Common Secure Interoperability (CSIv2) specification are documented with regard to the Security Attribute Service (SAS) protocol that allows for interoperable authentication, delegation and privileges.

A description of how a Java client should be configured to make use of the security features follows with a look at the options available.

Programmatic login is discussed next with examples.

6.1 Java clients

A client is a generic term used to refer to the process typically responsible for requesting a service. The service is provided by the server. A client container may be used to provide the necessary environment in which the client can issue a request for service.

With version 5, the Application Server consists of five application client models.

► ActiveX application client.

► Applet application client

► J2EE application client

► Pluggable application client

► Thin application client

The ActiveX application client makes use of the Java Native Interface (JNI) to provide programmatic access to the Java Virtual Machine (JVM). The JVM exists in the same process space as the ActiveX application, which may be written in Visual Basic, VBScript or Active Server Pages (ASP). Java objects contained in the JVM can be accessed, via a proxy, from the ActiveX application. Thus, using the J2EE client programming model, the ActiveX application client can gain access to EJB's residing in the Application Server. However, due to the nature of ActiveX applications, this model is only available on the Windows platform.

The applet application client makes use of a JVM embedded in the Web browser that the applet is running in. There are no tools provided to aid the programmer in developing the applet, generating the client-side bindings and deploying the code, although the applet application client will provide the runtime to support communication with the server. None-the-less, this model provides a lightweight client that can be readily downloaded and installed and there is no need to distribute the applet to the client machine since this operation is performed by the Web browser.

The J2EE application client operates in its own JVM which provides access to some J2EE APIs, such as JNDI, JDBC, RMI-IIOP and JMS. The application client is written in Java and relies on the application runtime to configure its environment. Tools are provided to aid the development, assembly, deployment and launching of a J2EE application client. Another benefit of this model is the use of short names in the deployment descriptor to identify remote and local resources.

The pluggable application client provides a lightweight Java client environment without the J2EE APIs. The pluggable application client requires access to the CosNaming or JNDI interfaces for EJB or CORBA object resolution. As with the applet application client, tools are not provided to aid development and installation. A non-IBM JRE must be installed and the pluggable application client must be distributed to the client machine.

The thin application client also provides a lightweight Java client environment in much the same way that the pluggable application client does. No tools are provided for the development and installation of the thin application client. Short names may not be used in the deployment descriptor to identify remote and local resources.

Shown below is a table listing the functionality of each of the application clients.

Table 6-1   Application client functions

Available functions	**ActiveX client**	**Applet client**	**J2EE client**	**Pluggable client**	**Thin client**
Provides all the benefits of a J2EE platform	Yes	No	Yes	No	No
Portable across all J2EE platforms	No	No	Yes	No	No
Provides the necessary runtime to support communication between client and server	Yes	Yes	Yes	Yes	Yes
Allows the use of short names in the deployment descriptor	No	No	Yes	No	No
Supports use of RMI-IIOP	Yes	Yes	Yes	Yes	Yes
Supports use of HTTP	Yes	Yes	No	No	No
Enables development of client apps that can access EJB references and CORBA object references	Yes	Yes	Yes	Yes	Yes

Available functions	ActiveX client	Applet client	J2EE client	Pluggable client	Thin client
Enables initialisation of client app's runtime environment	No	No	Yes	No	No
Supports authentication to local resources	No	No	Yes	No	No
Requires app is distributed to client machine	No	No	Yes	Yes	Yes

This chapter will concentrate on securing the J2EE application client and thin application client.

6.2 CSIv2 and SAS

The Common Secure Interoperability (CSI) security specification is defined by the OMG (see `http://www.omg.org`). Currently in its second version, the specification defines the Security Attribute Service (SAS) protocol to address the requirements of CORBA security for interoperable authentication, delegation and privileges. The SAS protocol is designed to exchange its protocol elements in the service context of GIOP request and reply messages that are communicated over a connection-based transport. The protocol is intended to be used in environments where transport layer security, such as that available via Secure Sockets Layer (SSL)/ Transport Layer Security (TLS) or Secure InterORB Protocol (SECIOP), is used to provide message protection (that is, integrity and or confidentiality) and server-to-client authentication. The protocol provides client authentication, delegation, and privilege functionality that may be applied to overcome corresponding deficiencies in an underlying transport. The SAS protocol facilitates interoperability by serving as the higher-level protocol under which secure transports may be unified.

The SAS protocol is divided into two layers:

► The authentication layer is used to perform client authentication where sufficient authentication could not be accomplished in the transport.

► The attribute layer may be used by a client to deliver security attributes, such as identity and privilege, to a target where they may 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 SAS 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 some identity other than itself. This can improve the performance of a system since the authentication of a client is relatively expensive. The server can validate the request by checking its trust rules.

In order to invoke an EJB method that has been secured, a protocol is required to determine the level of security and type of authentication to be agreed upon by the client and server. During the method invocation, the protocol must coalesce the server's authentication requirements, which is determined by the object's IOR, with the client's authentication requirements, which is determined by the client's configuration and select the appropriate policy.

The Application Server can be configured to support both CSIv2 and IBM's Secure Association Service (SAS). In fact, both protocols can be supported *simultaneously*; that is to say the Application Server may receive a request using one protocol and then receive another request using the other protocol. IBM's SAS is the protocol used in previous versions of the Application Server and although deprecated, is provided in version five for interoperability with older clients and servers. CSIv2, which is the focus of this chapter, allows vendors to securely interoperate and provides a greater number of features over SAS.

CSIv2 and SAS are *add-on* IIOP services, where IIOP is the communications protocol used to send messages between two ORBs. In preparation for a request to flow from client to server, a connection between the two ORBs must be established over TCP/IP. The client ORB will invoke the authentication protocol's client connection interceptor which is used to read the tagged components in the IOR of the server-based object being requested. This is how the authentication policy is established. Once the policy has been established, the ORB will make the connection, with the optional addition of the SSL cipher.

The client ORB invokes the client request interceptor once the connection has been established and sends security information other than what was established by the transport. This may include a user ID and password token, which is authenticated by the server, an authentication mechanism-specific token, which is validated by the server or an identity assertion token, which 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 service context.

Upon receiving the message, the server ORB invokes the authentication protocol's server request interceptor, which finds the service context added by the client's request interceptor and invokes a method in the Security server to authenticate the client's identity. If the client is authentic, the Security server will return a credential containing additional client information which it retrieved from the user registry in order to allow for authorisation decisions to be made when the EJB method corresponding to the client request is invoked.

Should the server's request interceptor find no service context, it will look at the transport connection information to see if a client certificate was supplied when the SSL connection between client and server was established. If such a certificate is found, the Distinguished Name (DN) is extracted and is mapped to an identity in the user registry. In the case of LTPA, the DN is used; for SWAM or Kerberos, the Common Name (CN) portion of the DN is used.

If identity information is not available, an unauthenticated credential will be created and applied in order to determine if the request is authorised to invoke the required method.
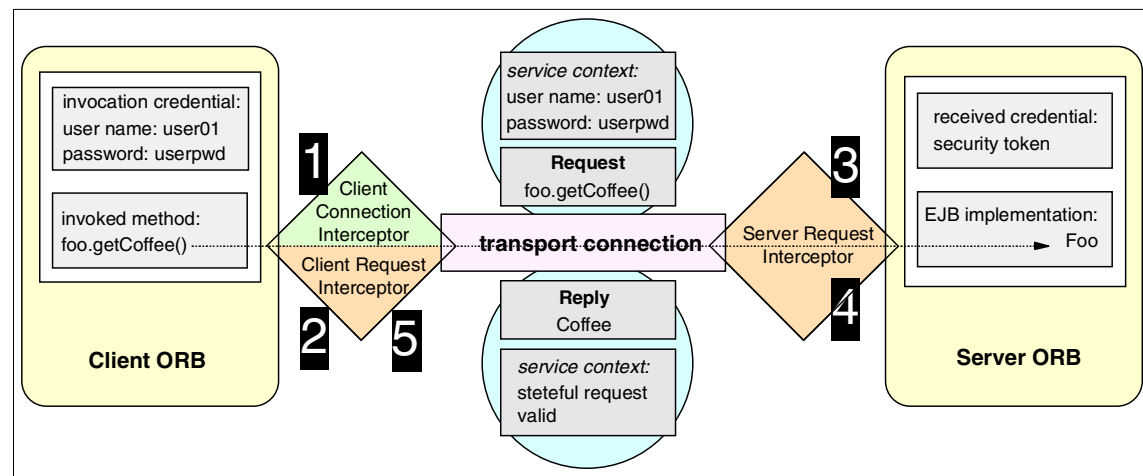
Figure 6-1   Authentication Protocol Flow

Follow the steps on the flow diagram above:

1. Client ORB calls the connection interceptor to create the connection.

2. Client ORB calls the request interceptor to get client security infromation ,send_request().

3. Server ORB calls the request interceptor to receive the security information, authenticate and set the received credential, receive_request().

4. Server ORB calls the request interceptor to allow security to send information back to the client along with the reply, send_reply().

5. Client ORB calls the request interceptor to allow the client to cleanup and set the session status as valid or invalid, receive_reply().

The SAS protocol supports the establishment of both *stateless* and *stateful* security contexts. Stateless contexts exist only for the duration of the GIOP request that was used to establish the context. Stateful contexts endure until they are discarded and can be referenced for use with subsequent requests. The SAS protocol includes a simple negotiation protocol that defines a least-common-denominator form of interoperability between implementations that support only stateless contexts and those that support both stateless and stateful forms. Whilst the J2EE specification requires that only stateless mode is implemented, the Application Server implements both in order to improve the performance of a multiple message application. This is achieved by invoking the server request interceptor when the EJB method has completed and sending a new reply service context to client containing the outcome. This extra step is only necessary for the first request in the session.
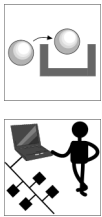
6.3 Configuring the Java client

It is necessary to configure the Java client in order that it can access secured applications. It must be made aware of certain properties, such as the security settings for the client ORB. The properties are provided in a file called sas.client.props. The JVM in which the application client will run should be set to use this property file by adding the directive: com.ibm.CORBA.ConfigURL=<URL_ of_the_properties_file>.

So, to start the JVM, enter:

```
java -Dcom.ibm.CORBA.ConfigURL=file:/<WAS_ROOT>/properties/sas.client.props
com.abc.myJavaApp
```

where <WAS_ROOT> should be replaced with the directory in which the Application Server was installed, for example `C:\WebSphere\AppServer` on a Windows machine.

The sas.client.properties file
The CORBA authentication options (with the valid values / default value in brackets) in the client property file are listed below.

► com.ibm.CORBA.securityEnabled (true, false / true) - determines if client security has been enabled

- ► com.ibm.CSI.protocol (ibm, csiv2, both / both) - determines which authentication protocol the client is permitted to use

- ► com.ibm.CORBA.authenticationTarget (BasicAuth) - determines type of authentication mechanism to use. The user name and password will be communicated to the server. In this case, SSL should be enabled in order to encrypt this information.

- ► com.ibm.CORBA.validateBasicAuth (true, false / true) - determines if the user details are authenticated immediately or deferred until the first method request is communicated to the server. Requires the com.ibm.CORBA.authenticationTarget property to be set to BasicAuth

- ► com.ibm.CORBA.authenticationRetryEnabled (true, false / true) - determines if a failed login should be retried. This also applies to stateful CSIv2 sessions and validations that have failed due to an expired credential. Only those failures which are known to be correctable will be retried

- ► com.ibm.CORBA.authenticationRetryCount (an integer within the range 1 and 10 / 3) - determines how many retries will be attempted. Requires com.ibm.CORBA.authenticationRetryEnabled to be set to true

- ► com.ibm.CORBA.loginSource (prompt, key file, stdin, none, properties / prompt) - determines how the authentication request interceptor will log in if it does not find a invocation credential set. Requires com.ibm.CORBA.loginUserid and com.ibm.CORBA.loginPassword properties to be set. Prompt will display a window requesting user name and password, key file will extract the user details from the file specified by com.ibm.CORBA.keyFileName, stdin will display a command line prompt requesting user details, none should be selected only if the client uses programmatic login (see Chapter 8, "Programmatic security" on page 179) and properties will retrieve the user details from the com.ibm.CORBA.loginUserid and com.ibm.CORBA.loginPassword properties

- ► com.ibm.CORBA.loginUserid (user ID / blank) - determines the user ID when the com.ibm.CORBA.loginSource property is set to properties. Requires com.ibm.CORBA.loginPassword property to be set and CSIv2 message layer authentication in use

- ► com.ibm.CORBA.loginPassword (password / blank) - determines the user password when the com.ibm.CORBA.loginSource property is set to properties. Requires com.ibm.CORBA.loginUserid property to be set and CSIv2 message layer authentication in use

- ► com.ibm.CORBA.keyFileName (path to keyfile / <WAS_INSTALL_ROOT>/properties/wsserver.key) - determines location of the key file that contains a list of realm/userid/password combinations. Used when the com.ibm.CORBA.loginSource property is set to key file

- ► com.ibm.CORBA.loginTimeout (an integer within the range 0 and 600 / 300) - determines the amount of time, in seconds, that the login prompt will be available before the login will be considered invalid

The SSL configuration options are listed below.

- ► com.ibm.ssl.protocol (SSL, SSLv2, SSLv3, TLS, TLSv1 / SSL) - determines which variety of the SSL and TLS protocols are used to perform transport-layer encryption
- ► com.ibm.ssl.keyStoreType (JKS, JCEK, PKCS12 / JKS) - determines the format of the SSL key store file
- ► com.ibm.ssl.keyStore (path to key store / <WAS_INSTALL_ROOT>/etc/DummyClientKeyFile.jks) - determines location of SSL key store file, which is used personal certificates and private keys. Dummy client and server key stores files are provided to aid development of applications that use key stores, without the need to generate keys or create a Certification Signing Request (CSR)
- ► com.ibm.ssl.keyStorePassword (the key store password / default password for DummyClientKeyFile.jks) - determines the password with which the key store file is protected
- ► com.ibm.ssl.trustStoreType (JKS, JCEK, PKCS12 / JKS) - determines the format of the SSL key trust file
- ► com.ibm.ssl.trustStore (path to trust store / <WAS_INSTALL_ROOT>/etc/DummyClientTrustFile.jks) - determines location of SSL key trust file
- ► com.ibm.ssl.trustStorePassword (the key trust password / default password for DummyClientTrustFile.jks) - determines the password with which the key trust file is protected
- ► com.ibm.CORBA.standardClaimQOPModels (low, medium, high / high) - determines the quality of protection (in other words, the security level). If the server and client values differ then the highest value will be chosen and the connection will be initialised with this QOP property. A list of supported ciphers for each level of QOP are provided in the InfoCenter

The CSIv2 configuration properties are listed below. Certain security properties have supported/required property pairs. If the required property is enabled then communication with the server must satisfy this property.

- ► com.ibm.CSI.performStateful (true, false / true) - determines if the authentication request should result in a stateful reply returning from the server
- ► com.ibm.CSI.performTLClientAuthenticationRequired (true, false / false) and com.ibm.CSI.performTLClientAuthenticationSupported (true, false / false) -

determines if transport-layer client authentication is required or supported. This will involve the client sending a digital certificate to the server during the authentication stage. If the Required property is set to true, the client will only authenticate with servers that support transport-layer client authentication

► com.ibm.CSI.performTransportAssocSSLTLSRequired (true, false / false) and com.ibm.CSI.performTransportAssocSSLTLSSupported (true, false / true) - determines if the client can use SSL to communicate with the server. If the Required property is set to true, the client will only communicate with servers that support SSL

► com.ibm.CSI.performClientAuthenticationRequired (true, false / true) and com.ibm.CSI.performClientAuthenticationSupported (true, false / true) - determines if message layer client authentication is required or supported. The com.ibm.CORBA.authenticationTarget property determines the type of authentication mechanism

► com.ibm.CSI.performMessageIntegrityRequired (true, false / true) and com.ibm.CSI.performMessageIntegritySupported (true, false / true) - determines if a connection secured by a 40-bit cipher is supported or required. If the Required property is set to true then the connection will fail if the server does not support 40-bit ciphers. This property is only valid when SSL is enabled

► com.ibm.CSI.performMessageConfidentialityRequired (true, false / false) and com.ibm.CSI.performMessageConfidentialitySupported (true, false / true) - determines if a connection secured by a 128-bit cipher is supported or required. If the Required property is set to true then the connection will fail if the server does not support 128-bit ciphers. This property is only valid when SSL is enabled.

For a more complete list of directives refer to the WebSphere Application Server InfoCenter for more details.

The Application Server should also be configured to communicate with a client in the required fashion. If a Java client requires that client certificates are transmitted via SSL, for example, then the server must be set to expect this. Details on the configuration of the Application Server can be found in Chapter 10, "Administering WebSphere security" on page 233.
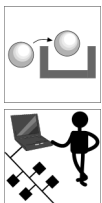
6.4 Identity Assertion

Definition: *Identity assertion* is basically the process when the invocation credential is asserted to the downstream server during a call.

When a client authenticates to a server, the received credential is set. When authorization checks the credential to see if it is allowed access, it will also set the invocation credential so that if the EJB method calls another EJB method located on other servers, the invocation credential can be used as the identity to invoke the downstream method. Depending on the RunAs mode for the EJB, the invocation credential will be set as the originating client identity, the server's identity, or a specified different identity. Regardless of the identity that is set, when Identity Assertion is enabled, it is the invocation credential that is asserted to the downstream server.

The invocation credential identity is sent to the downstream server in an identity token. In addition, the sending server's identity including password or token, is sent in the client authentication token. Both are needed by the receiving server to accept the asserted identity. The receiving server does the following to accept the asserted identity.

1. First it is determined if the sending server's identity is on the trusted principal list of the receiving server. That is, is the sending server one which is allowed to send an identity token to the receiving server.

2. Second, once we have determined that the sending server is on the trusted list, we need to make sure it truly is the sending server by authenticating it. This could be simply comparing the user ID and password from the sending server to that of the receiving server. Or it could require a real authenticate call.

3. If the sending server's credentials are authenticated and on the trusted principal list, then evaluation of the identity token can proceed. Evaluation of the identity token consists of the following. There are four formats of identities which can be present in an identity token:

   – principal name

   – distinguished name

   – certificate chain

   – anonymous identity

   The WebSphere Application Servers that receive authentication information typically support all four identity types. The sending server decides which one will be chosen based on how the original client authenticated. The type that is present depends on how the client originally authenticates to the sending server. For example, if the client uses SSL client authentication to authenticate to the sending server, then the identity token to the downstream server will contain the certificate chain. This is important because it allows the receiving server to perform it's own mapping of the certificate chain. It enables more interoperability with other vendors and platforms.

4. Once the identity format is understood and parsed, the identity is simply mapped to a credential. All identity token types map to the user ID field of the active user registry. It is done by mapping Distinguished Name to Distinguished Name and using filters to allow administrators to control the mapping.

5. Some user registry methods are called to gather additional credential information used by authorization. In a stateful server, this is done one time for the sending server/receiving server pair where the identity tokens are the same. Subsequent requests will be made via a session ID.

> **Note:** Identity Assertion is only available using the CSIv2 protocol.

6.4.1 Scenarios

The following sections will describe five different cases where identity assertion is utilized to propagate credentials to downstream servers.

Sample application for the scenarios

A testing application is also distributed with this book, besides the ITSOBank sample application. It is the Identity Assertion sample application made for testing purposes for the following scenarios (Scenario 1, 2, 3 and 4).

The application is very simple, it consists three modules:

► a J2EE client,

► an EJB module for the front-end server,

► an EJB module for the back-end server.

The application does the following:

1. The client sends a message to the front-end server.

2. On the front-end server the PassThrough session EJB captures the message, then passes the message to the back-end server.

3. On the back-end server the Bouncer session EJB captures the message, attaches its own message with the caller's user name, and returns the message to the front-end server.

4. The PassThrough session EJB gets the response, attaches its own message together with the caller's user name, client is the caller.

5. The client gets the response back from the front-end server and dumps it out to the standard out.
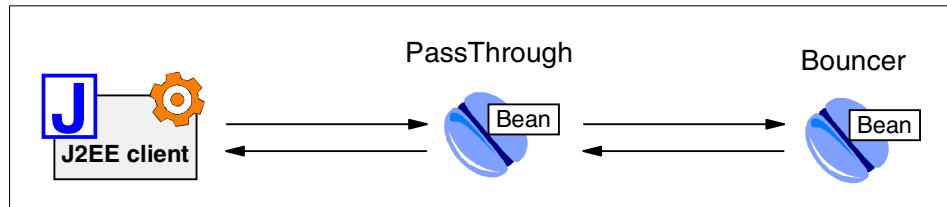
Figure 6-2   Identity Assertion application

There is a .properties file for the application, that stores the information for the EJB clients, where to find the EJB server. It is stored in the IDAssertionCommon.jar file, called: *ejblocations_en_US.properties*. You can modify the settings using the Application Assembly Tool, WebSphere Studio or editing the file in the deployed application. Specify the following properties for the application, for example:

```
ejb.front.server.hostname=server01
ejb.front.server.port=2809
ejb.back.server.hostname=server02
ejb.back.server.port=2809
```

Leave the bean names unchanged.

The sample application is running in a three machine environment, scenario 2 might require a second client machine, unless you re-configure the first client.

In this environment the client machine is running the WebSphere Client runtime, installed form the WebSphere Client CD, while the other two systems are running the WebSphere Application Server base server.
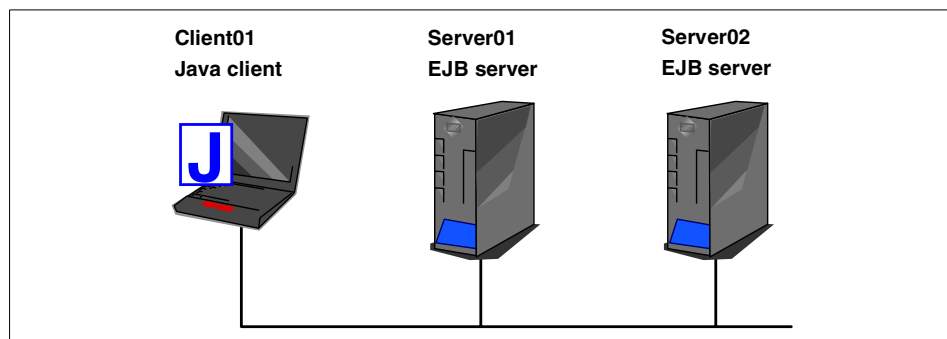
Figure 6-3   Test environment for Identity Assertion

Scenario 1: BasicAuth and Identity Assertion

This is an example of a pure java client, Client01, accessing a secure EJB on Server01 via user "user01". The EJB code on Server01 accesses another EJB on Server02. This configuration uses Identity Assertion to propagate the identity of "user01" to the downstream server Server02. Server02 will trust that "user01" has already been authenticated by Server01 because it trusts Server01. To gain this trust, the identity of Server01 also flows to Server02 simultaneously and Server02 will validate the identity by checking the trustedPrincipalList to ensure it is a valid server principal. Server02 also authenticates Server01.
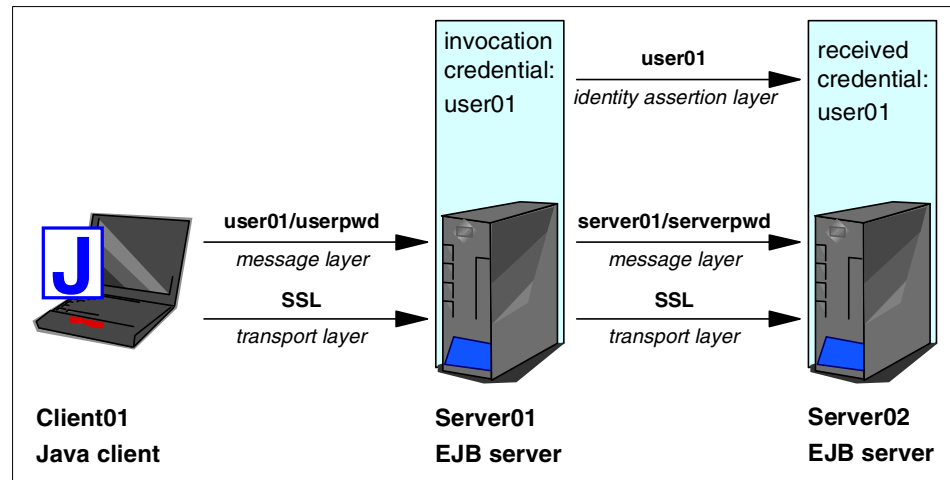
Figure 6-4   Scenario 1: BasicAuth and Identity Assertion

The following shows the steps for configuring Client01, Server1 and Server2.

Configuring Client01

Client01 requires message layer authentication with an SSL transport, follow the steps to configure Client01.

1. The client needs to point to the sas.client.props file using the property in the command line, see *Section 6.3, "Configuring the Java client" on page 103* for more information on this; you can use the following parameter: *com.ibm.CORBA.ConfigURL=file:/c:/websphere/appclient/properties/sas.clie nt.props*.

2. All further configuration involves setting properties within the sas.client.props file, open it in a text editor at the <WEBSPHERE_CLIENT_ROOT>/properties directory.

3. Enable SSL for the connection, in this case, SSL will be supported but not required: *com.ibm.CSI.performTransportAssocSSLTLSSupported=true*, *com.ibm.CSI.performTransportAssocSSLTLSRequired=false*.

4. Enable client authentication at the message layer. In this case, client authentication is supported but not required:
com.ibm.CSI.performClientAuthenticationRequired=false,
com.ibm.CSI.performClientAuthenticationSupported=true.

5. Use all of the rest of the defaults in the sas.client.props file. Save the file then close it.

Configuring Server01

In the Web Console, Server01 will be configured for incoming requests to support message layer client authentication and incoming connections to support SSL without client certificate authentication. Server01 will be configured for outgoing requests to support identity assertion. Follow the steps to configure Server01:

1. Configure Server01 for incoming connections. Start the Administrative Console for Server01, then navigate to the **Security -> Authentication Protocol** section.

   a. Select **CSIv2 Inbound Authentication.**

      i. Enable Basic Authentication, by selecting **Supported**.

      ii. Disable Client Certificate Authentication by selecting **Never**.

      iii. Disable Identity Assertion.

   b. Select **CSIv2 Inbound Transport**.

      i. Enable SSL, by selecting **SSL-Supported**.

2. Configure Server01 for outgoing connections.

   a. Select **CSIv2 Outbound Authentication.**

      i. Disable Basic Authentication, by selecting **Never**.

      ii. Disable Client Certificate Authentication by selecting Never.

      iii. Enable Identity Assertion.

   b. Select **CSIv2 Outbound Transport**.

      i. Enable SSL, by selecting **SSL-Supported**.

Configuring Server02

In the Web Console, Server02 will be configured for incoming requests to support identity assertion and to accept SSL connections. Complete the following to configure incoming connections.

Configuration for outgoing requests and connections are not relevant for this scenario. Follow the steps to configure Server02:

1. Configure Server02 for incoming connections. Start the Administrative Console for Server02, then navigate to the **Security -> Authentication Protocol** section.

2. Select **CSIv2 Outbound Authentication.**

   a. Disable Basic authentication, by selecting **Never**.

   b. Disable Client Certificate Authentication by selecting **Never**.

   c. Enable Identity Assertion.

3. Select **CSIv2 Outbound Transport**.

   a. Enable SSL, by selecting **SSL-Supported**.

Testing the scenario

To test this scenario, simply launch the Identity Assertion J2EE clinet on the client machine using the following command:

```
launchClient IDAssertion.ear
```

The client should be configured already to know where to find the WebSphere Application Server, server01.

> **Note:** When you installed the Client Runtime, WebSphere asked for the server name and the port number.

Provide the username, password and realm name when the client asks for it.

After running the client, you should see the messages from the different servers in your console. Check if you see the message from the PassThrough bean, and from the Bouncer bean, together with the caller user names.

You can also turn on tracing for the WebSphere Application Servers then check the trace file and see what happened during the process.

Scenario 2: BasicAuth, Identity Assertion and Client Certificates

This scenario is the same as Scenario 1 except for the interaction from client Client02 to server Server02. Therefore, the configuration of Scenario 1 still needs to be in place, but we have to modify server Server02 slightly and add a configuration for client Client02. We will not be modifying the configuration for Client01 or Server01; follow the steps from Section , "Scenario 1: BasicAuth and Identity Assertion" on page 110.
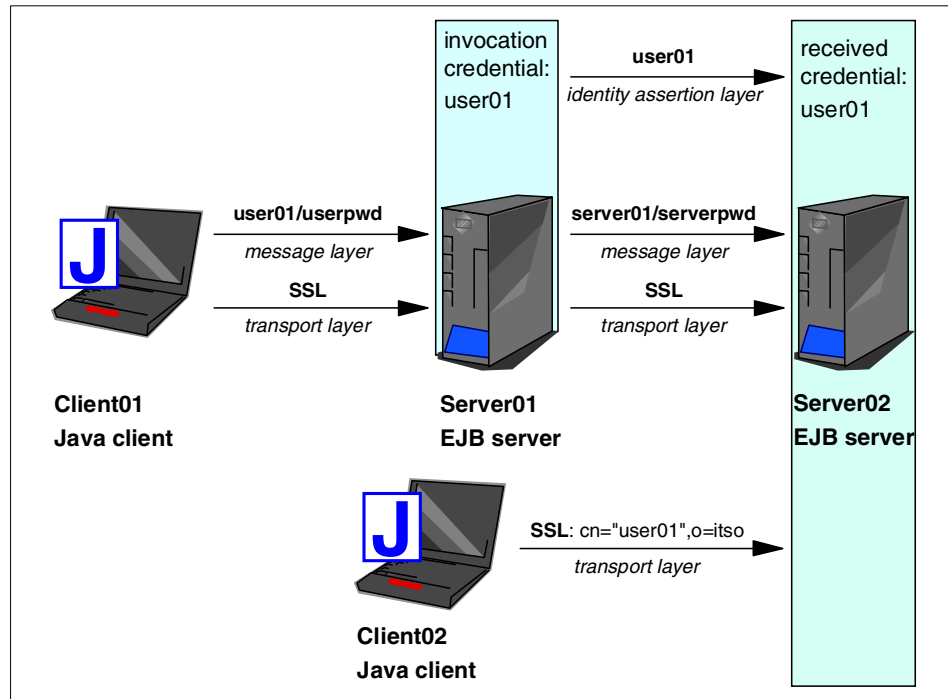
Figure 6-5   Scenario 2: BasicAuth, Identity Assertion and Client certificates

Configuring Client02

Client02 requires transport layer authentication using SSL client certificates, to accomplish this follow the steps below:

1. The client needs to point to the sas.client.props file using the property *com.ibm.CORBA.ConfigURL=file:/c:/websphere/appclient/properties/sas.client.props*.

2. All further configuration involves setting properties within the sas.client.props file, open it in a text editor at the <WEBSPHERE_CLIENT_ROOT>/properties directory.

3. Enable SSL fo the connection, in this case, SSL will be supported but not required: *com.ibm.CSI.performTransportAssocSSLTLSSupported=true, com.ibm.CSI.performTransportAssocSSLTLSRequired=false*.

4. Disable client authentication at the message layer. *com.ibm.CSI.performClientAuthenticationRequired=false, com.ibm.CSI.performClientAuthenticationSupported=false*.

5. Enable client authentication at the transport layer. Here we are supporting it and not requiring it:

com.ibm.CSI.performTLClientAuthenticationRequired=false,
com.ibm.CSI.performTLClientAuthenticationSupported=true.

6. Save the file then close it.

Configuring Server02

In the Web Console, Server02 will be configured for incoming requests to SSL client authentication and Identity Assertion. Configuration for outgoing requests is not relevant for this scenario. Follow the steps below to configure Server02.

Configure Server02 for incoming connections.

1. Configure Server02 for incoming connections. Start the Administrative Console for Server02, then navigate to the **Security -> Authentication Protocol** section.

2. Select **CSIv2 Incoming Authentication.**

    a. Disable Basic authentication, by selecting **Never**.

    b. Enable Client Certificate Authentication by selecting Supported.

    c. Enable Identity Assertion.

3. Select **CSIv2 Incoming Transport**.

    a. Enable SSL, by selecting **SSL-Supported**.

> **Note:** An important concept to grasp is that these configuration options can be mixed and matched. However, there is a precedence to which authentication features will become the identity in the received credential. The order of precedence is as follows:
>
> 1. Identity assertion
> 2. Message layer client authentication (BasicAuth or token)
> 3. Transport layer client authentication (SSL certificates)

Scenario 3: Client certificate and RunAs system

This is an example of a pure java client, Client01, accessing a secure EJB on Server01. Client01 authenticates to Server01 using SSL client certificates. Server01 maps the cn of the DN in the certificate to a user in the local registry. The user in this case will be "user01". The EJB code on Server01 accesses another EJB on Server02. Because the RunAs mode is system, the invocation credential is set as "Server01" for any outbound requests.
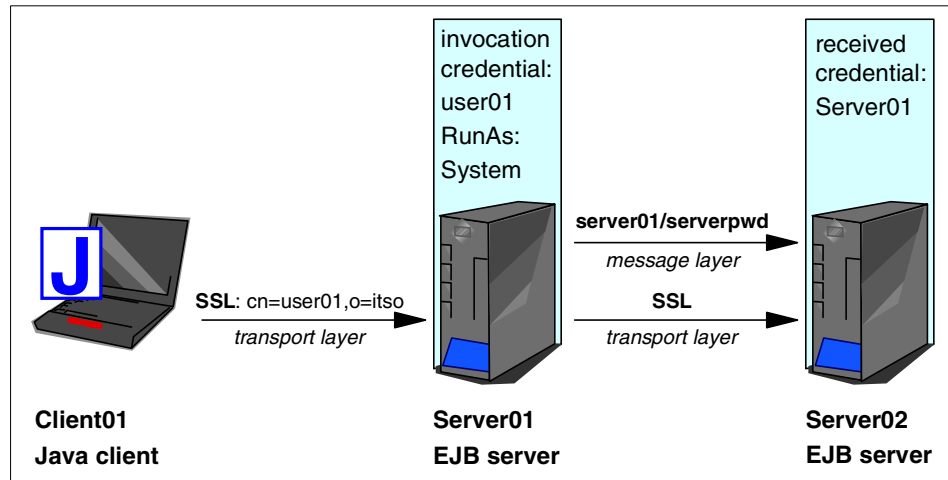
Figure 6-6   Scenario 3: Client certificate and RunAs system

Configuring Client01

Client01 requires transport layer authentication (SSL client certificates). Follow the steps below to configure Client01.

1. The client needs to point to the sas.client.props file using the property *com.ibm.CORBA.ConfigURL=file:/c:/websphere/appclient/properties/sas.clie nt.props*.

2. All further configuration involves setting properties within the sas.client.props file, open it in a text editor at the <WEBSPHERE_CLIENT_ROOT>/properties directory.

3. Enable SSL for the connection, in this case, SSL will be supported but not required: *com.ibm.CSI.performTransportAssocSSLTLSSupported=true, com.ibm.CSI.performTransportAssocSSLTLSRequired=false*.

4. Disable client authentication at the message layer. *com.ibm.CSI.performClientAuthenticationRequired=false, com.ibm.CSI.performClientAuthenticationSupported=false*.

5. Enable client authentication at the transport layer. Here we are supporting it and not requiring it: *com.ibm.CSI.performTLClientAuthenticationRequired=false, com.ibm.CSI.performTLClientAuthenticationSupported=true*.

6. Save the file then close it.

Configuring Server01

In the Web Console, Server01 will be configured for incoming connections to support SSL with client certificate authentication. Server01 will be configured for outgoing requests to support message layer client authentication. Follow the steps below to configure Server01:

1. Configure Server01 for incoming connections. Start the Administrative Console for Server01, then navigate to the **Security -> Authentication Protocol** section.

   a. Select **CSIv2 Inbound Authentication.**

      i. Disable Basic Authentication, by selecting **Never**.

      ii. Enable Client Certificate Authentication by selecting **Supported**.

      iii. Disable Identity Assertion.

   b. Select **CSIv2 Inbound Transport**.

      i. Enable SSL, by selecting **SSL-Supported**.

2. Configure Server01 for outgoing connections.

   a. Select **CSIv2 Outbound Authentication.**

      i. Disable Basic Authentication, by selecting **Never**.

      ii. Enable Client Certificate Authentication by selecting **Supported**.

      iii. Disable Identity Assertion.

   b. Select **CSIv2 Outbound Transport**.

      i. Enable SSL, by selecting **SSL-Supported**.

Configuring Server02

In the Web Console, Server02 will be configured for incoming requests to support message layer authentication over SSL. Configuration for outgoing requests is not relevant for this scenario. Follow the steps below to configure Server02:

1. Configure Server02 for incoming connections. Start the Administrative Console for Server02, then navigate to the **Security -> Authentication Protocol** section.

2. Select **CSIv2 Outbound Authentication.**

   a. Enable Basic authentication, by selecting **Supported**.

   b. Disable Client Certificate Authentication by selecting **Never**.

   c. Disable Identity Assertion.

3. Select **CSIv2 Outbound Transport**.

   a. Enable SSL, by selecting **SSL-Supported**.

Scenario 4: TCP/IP Transport using VPN

This scenario illustrates the ability to choose TCP/IP as the transport when it's appropriate to do so. In some cases, when two servers are on the same VPN, it may be appropriate to select TCP/IP as the transport for performance reasons since the VPN already encrypts the message.
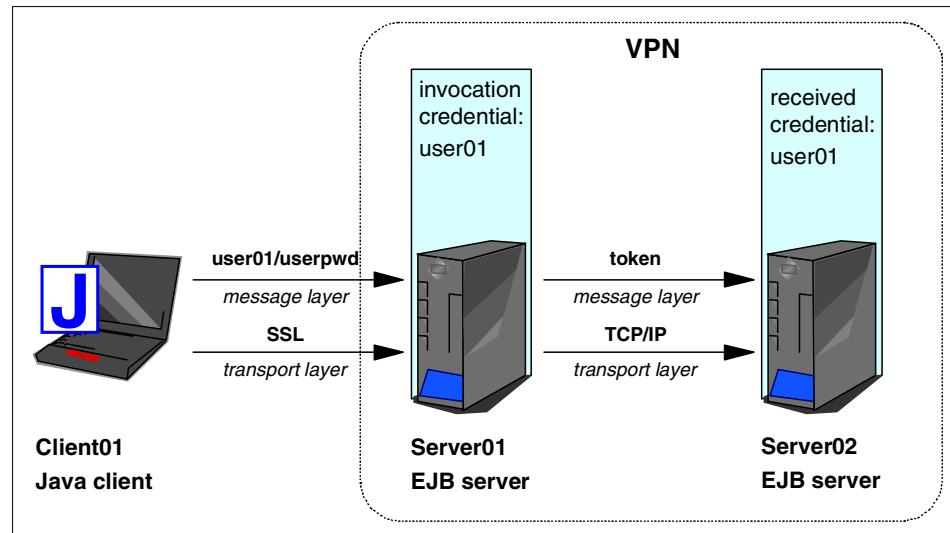
Figure 6-7   Secnario 4: TCP/IP Transport using VPN

Configuring Client01

Client01 requires message layer authentication with an SSL transport. follow the steps below to configure Client01.

1. The client needs to point to the sas.client.props file using the property *com.ibm.CORBA.ConfigURL=file:/c:/websphere/appclient/properties/sas.client.props*.

2. All further configuration involves setting properties within the sas.client.props file, open it in a text editor at the <WEBSPHERE_CLIENT_ROOT>/properties directory.

3. Enable SSL for the connection, in this case, SSL will be supported but not required: *com.ibm.CSI.performTransportAssocSSLTLSSupported=true, com.ibm.CSI.performTransportAssocSSLTLSRequired=false*.

4. Enable client authentication at the message layer. In this case, client authentication is supported but not required: *com.ibm.CSI.performClientAuthenticationRequired=false, com.ibm.CSI.performClientAuthenticationSupported=true*.

5. Save the file then close it.

Configuring Server01

In the Administrative console, Server01 will be configured for incoming requests to support message layer client authentication and incoming connections to support SSL without client certificate authentication. Server01 will be configured for outgoing requests to support identity assertion. Follow the steps below to configure Server01:

1. Configure Server01 for incoming connections. Start the Administrative Console for Server01, then navigate to the **Security -> Authentication Protocol** section.

   a. Select **CSIv2 Inbound Authentication.**

      i.   Enable Basic Authentication, by selecting **Supported**.

      ii. Disable Client Certificate Authentication by selecting **Never**.

      iii. Disable Identity Assertion.

   b. Select **CSIv2 Inbound Transport**.

      i.   Enable SSL, by selecting **SSL-Supported**.

2. Configure Server01 for outgoing connections.

   a. Select **CSIv2 Outbound Authentication.**

      i.   Enable Basic Authentication, by selecting **Supported**.

      ii. Disable Identity Assertion.

   b. Select **CSIv2 Outbound Transport**.

      i.   Disable SSL, by selecting **TCPIP**.

Note: It is possible to enable SSL for inbound connections and disable SSL for outbound connections. The same is true in reverse.

Configuring Server02

In the Administrative onsole, Server02 will be configured for incoming requests to support identity assertion and to accept SSL connections. Configuration for outgoing requests and connections are not relevant for this scenario. Follow the steps below to configure Server02:

1. Configure Server02 for incoming connections. Start the Administrative Console for Server02, then navigate to the **Security -> Authentication Protocol** section.

2. Select **CSIv2 Outbound Authentication.**

   a. Enable Basic authentication, by selecting **Supported**.

   b. Disable Client Certificate Authentication by selecting **Never**.

     c. Disable Identity Assertion.

3. Select **CSIv2 Outbound Transport**.

     a. Disable SSL, by selecting **TCPIP**.

Scenario 5: Interoperability with WebSphere Application Server 4.x

The purpose of this scenario is to show how secure interoperability can take place between different releases simultaneously while using multiple authentication protocols (SAS and CSIv2). For a WebSphere V5 server to communicate with a WebSphere V4 server, the WebSphere V5 server must support either IBM or BOTH as the protocol choice. By choosing BOTH, that WebSphere V5 server can also communicate with other WebSphere v5 servers which support CSI. If the only servers in your security domain are WebSphere V5, it is recommended to choose CSI as the protocol since this will prevent the IBM interceptors from loading. However, if there's a chance that any server will need to communicate with a previous release of WebSphere, select the protocol choice of BOTH.
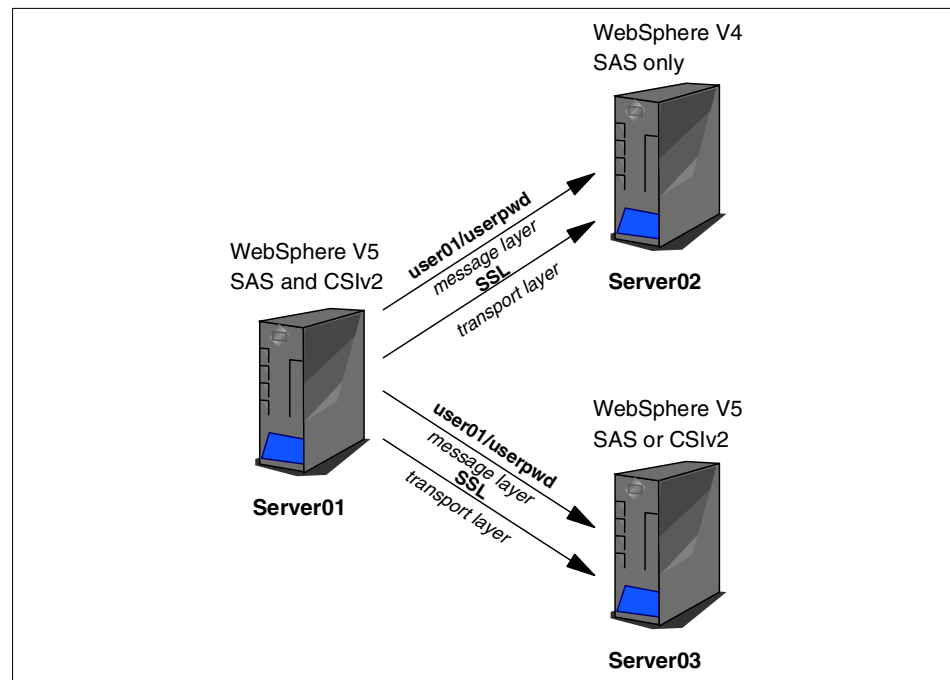
Figure 6-8   Scenario 5: Interoperability with WebSphere Application Server 4.x

Configuring Server01

Server01 requires message layer authentication with an SSL transport. Also, the protocol for Server01 must be BOTH. Configuration for incoming requests for Server01 is not relevant for this scenario. Follow the steps below to Configure Server01.

1. Configure Server01 for outgoing connections. Start the Administrative Console for Server01, then navigate to the **Security -> Authentication Protocol** section.

2. Select Global Security.

   a. Select **CSI and SAS** for active protocol.

3. Select **CSIv2 Outbound Authentication.**

   a. Enable Basic Authentication, by selecting **Supported**.

   b. Disable Client Certificate Authentication by selecting **Never**.

   c. Disable Identity Assertion.

4. Select **CSIv2 Outbound Transport**.

   a. Enable SSL, by selecting **SSL-Supported**.

Configuring Server02

All previous releases of WebSphere Application Server only support the SAS authentication protocol. There are no special configuration steps needed other than enabling global security on server (Server02).

Configuring Server03

In the Administrative console, Server03 will be configured for incoming requests to message layer authentication and to accept SSL connections. Configuration for outgoing requests and connections are not relevant for this scenario. follow the steps below to configure Server03.

1. Configure Server03 for outgoing connections. Start the Administrative Console for Server03, then navigate to the **Security -> Authentication Protocol** section.

2. Select Global Security.

   a. Select **CSI and SAS** or **CSI** for active protocol.

3. Select **CSIv2 Outbound Authentication.**

   a. Disable Basic Authentication, by selecting **Never**.

   b. Disable Client Certificate Authentication by selecting **Never**.

   c. Enable Identity Assertion.

4. Select **CSIv2 Outbound Transport**.

a. Enable SSL, by selecting **SSL-Supported**.

> **Note:** If Server03 is only communicating with WebSphere Application Server Version 5.0 servers, it is recommended to choose the CSI protocol. Otherwise, choose a protocol of BOTH.

6.5 J2EE application client

A J2EE application client operates in a similar fashion to a J2EE server-based application and has access to some of the J2EE API's. The application client will perform a JNDI lookup to retrieve a home interface as per usual J2EE programming practices. The process for developing and assembling a J2EE application client is documented in the InfoCenter.

The ITSOBank application provided with this book has a J2EE application client which requests service from an EJB operating in the remote EJB container. The EJB's home interface is called com.ibm.itsobank.ejbs.ConsultationHome which the client accesses be querying the JNDI namespace. The source code for the server and client components is included in the application EAR file. The client does not contain any code for logging in since this mechanism is provided by the client runtime environment.

The *launchclient* command (which is located in the <WAS_INSTALL_ROOT>/bin directory) will configure the J2EE application client environment by examining the application client's deployment descriptor (application-client.xml) which is provided with the application client. The client runtime environment must provide access to the EJB JAR files that contain the classes used by the client. These JAR files should be referenced in the client's MANIFEST.MF file. The recommendation is to provide the application EAR file, which will contain the classes for the entire application.

ITSOBank J2EE client

The ITSOBank application uses a Java application with a graphical interface to retrieve balance information for the accounts. The following steps describe how to start the application client in order to access the secured resources.

1. Use the following command to launch the J2EE client.

```
launchclient itsobank.ear -CCBootstrapHost=<AppServer_hostname>
-CCBootstrapPort=<AppServer_port>
```

If you do not specify host name and port number the client will connect to the *localhost* using the default port *2809*.

2. If Global Security has been enabled, a window should appear requesting user details (see Figure 6-9 on page 122).
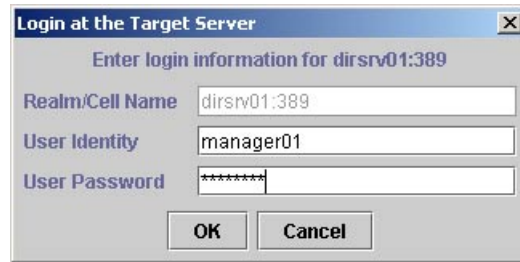
Figure 6-9   J2EE client challenge

3. Enter the user name and password of a user in the consultant group.

4. Click **OK.**

Once the client has been authenticated, the appropriate remote method will be invoked.

The client runtime determines which method to use to collect the user details by checking the com.ibm.CORBA.loginSource property in the sas.client.props file. The default value is prompt which causes the window to appear. However, changing this to stdin will cause the client runtime to request the details on the command line.

If the wrong user details are entered four times, the Application Server will throw a `javax.naming.NoPermissionException` exception with a `com.ibm.websphere.security.auth.AuthenticationFailedException` as the reason.

6.6 Java thin application client

The thin application client phrase refer to the Java client that is not running within the J2EE client container. It is a standalone Java application, that implements EJB clients connecting to an EJB container of WebSphere Application Server.

The clients usually run on a client machine separated from the application server. In order to connect to the server you have to provide the connection information for the EJB client, including the server name and the port number (the default port number for WebSphere Application Server V5 is 2809).

There are certain configurations you have to set for the JVM in order to operate in a secure environment. These settings are the followings:

► Djava.security.auth.login.config - tells the JVM where to find the JAAS login configuration file. For example:

```
Djava.security.auth.login.config=file:properties/wsjaas_client.conf
```

▶ Dcom.ibm.CORBA.ConfigURL - points to the file containing the client SAS settings for IBM SAS and CSIv2. For example:

```
Dcom.ibm.CORBA.ConfigURL=file:properties/sas.client.props
```

There are libraries also you have to include in your classpath when running a Java thin client in a secure environment. You will need the following .jar files from the WebSphere library at <WebSphere_root>/libs:

▶ wssec.jar

▶ namingclient.jar

▶ lmproxy.jar

▶ sas.jar

▶ ecutils.jar

As an additional library do not forget to add the EJB client code, home and remote classes of the EJB, to your classpath.

In your thin Java application you can leave the authentication challenge to the WebSphere code; or you can program your own login module with your own callback implementation.

The libraries included in the classpath contain the necessary code for the client to perform the authentication for WebSphere. The default source for login is the GUI login panel, you can change it to something else in the SAS client configuration file.

For more information about the programmatic login for thin Java clients refer to 8.7.2, "Client-side login with JAAS" on page 209.

Unzip the itsobankThinClient.zip file to a directory of your choice. You will find the source and compiled classes in the directory. There are also supporting properties files and key files for the secure connection.

In order to run the client, use the following syntax:

```
runclient server_name server_port [login]
```

The *server_name* and the *server_port* defines the remote application server location for the remote EJB connection. The optional *login* parameter tells the application to use programmatic login instead of the built in login mechanism.

The following example runs the application using WebSphere's login challenge:

```
runclient appsrv01 2809
```

Chapter 6. Securing Java clients   **123**

6.7 Where to find more information

For more information on the security aspects of J2EE, see the following documents:

► The Java 2 Platform Specification v1.3 at
  http://java.sun.com/j2ee/docs.html

► The Object Management Group (OMG) specification about CSIv2 at
  http://www.omg.org/technology/documents/formal/omg_security.htm#CSIv
  2

7

Securing Enterprise Integration components

This chapter consists of the discussion of three enterprise integration components security:

► Web services

► Messaging Services

► Java 2 Connectors (J2C)

These are all major areas in the context of application server, however from the security point of view this book will discuss them under one chapter.

7.1 Web Services security

Web Services became a hot area in relation with Web applications, it is a fairly new technology with remarkable promises. There are areas in Web Services yet to be solved or refined; like security, it is an essential part of Web Services, but the recommendations have been just worked out recently and it will take time to get mature.

This book will not introduce the concept Web Services, and it will not discuss the Web Services architecture; for a good introduction and more deatils on this topic read the IBM Redbook: Web Services Wizardry with WebSphere Studio Application Developer SG24-6292.

In this section you will find information, how to create a secured Web Service using digital certificates in WebSphere Studio. You will find an introduction to the WS-Security recommendation, that provides the security fundamentals for Web Services. At the end a few sections will discover how the Web Services Gateway is involved in Web Services security.

7.1.1 Digital Certificates

Digital signatures provide integrity, signature assurance and non-repudiation over Web data. Such features are especially important for documents that represent commitments such as contracts, price lists, and manifests. In view of recent Web technology developments, the proposed work will address the digital signing of documents (any Web resource addressable by a URI) using XML syntax. This capability is critical for a variety of electronic commerce applications, including payment tools.

Developing secure Web Services with WebSphere Studio

Now we are going to write a secured Web service to provide customer account balance details.

There is one pattern of evolving Web Services from an existing Web Server applications known as "Browser to Server Pattern". This pattern wraps an existing application as a service using a SOAP message as the service invocation. The Web server provides a runtime execution container that definies its own security model with policy information derived from a deployment descriptor configured by the deployer of the Web server application. This pattern typically includes a mechanism for associating the identity of the invoking entity (the browser client) with the executing application instance and allows the application to continue to function as it did before.

We already have a ConsultationHelper class that provides this information and we are going to convert this as the Web Service.

Generating the secure Web Service

In this section, a new secure Web Service will be added to the ITSOBank sample application; for development WebSphere Studio Application Developer(WSAD) V5.0 is used.

1. Select to the **J2EE perspective**, switch to the **J2EE Navigator** and select the **ConsultationHelper.java** as indicated on the picture below Figure 7-1.
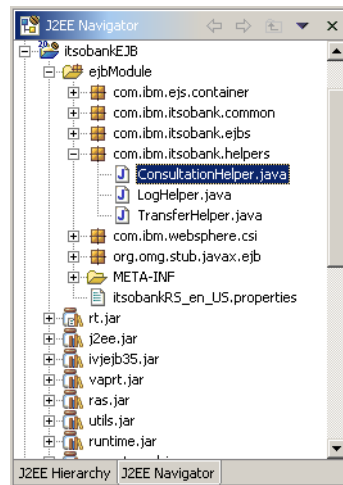
Figure 7-1   ConsultationHelper.java

2. Developers can look for this class in the package com.ibm.itsobank.helpers. You might want to have a look at the code for this ConsultaionHelper class before we generate a secured Web service for this component.

3. Select **File -> New -> Other** from the menu to create a new Web services in WebSphere Studio.
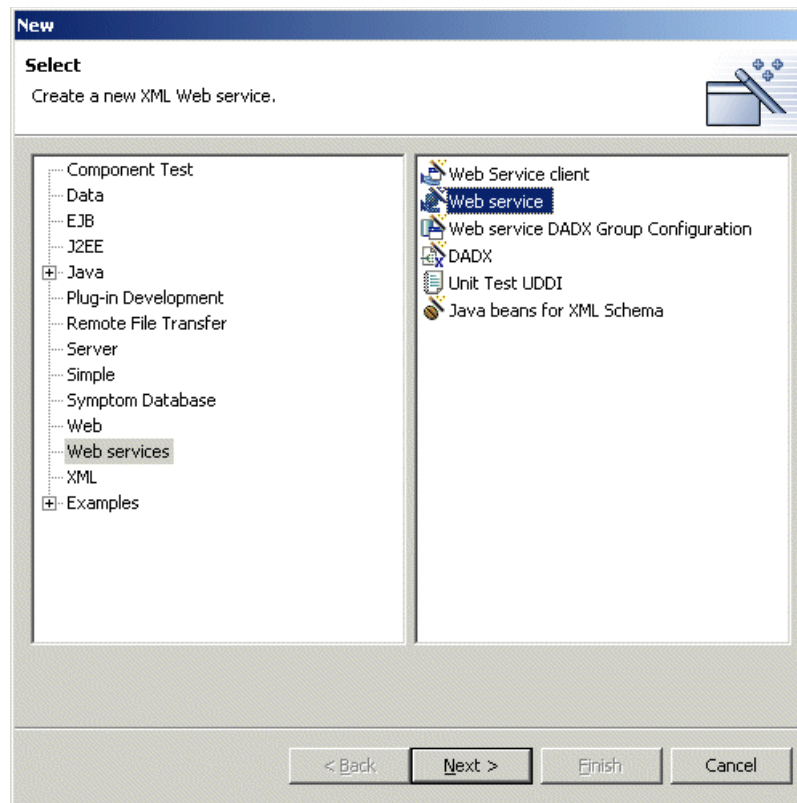
Figure 7-2   Creating a new Web Service

4. Select **Web Services** in the left-hand pane, then select **Web Services** in the right-hand pane. Click **Next** to start the Web services wizard.

Figure 7-3   Selecting Web Service type and client proxy type

5. In the **Service** section for Web service type select **Java bean Web servcie** from the drop-down list. Also click **Generate a proxy** checkbox and **Generate a sample**. click **Next**
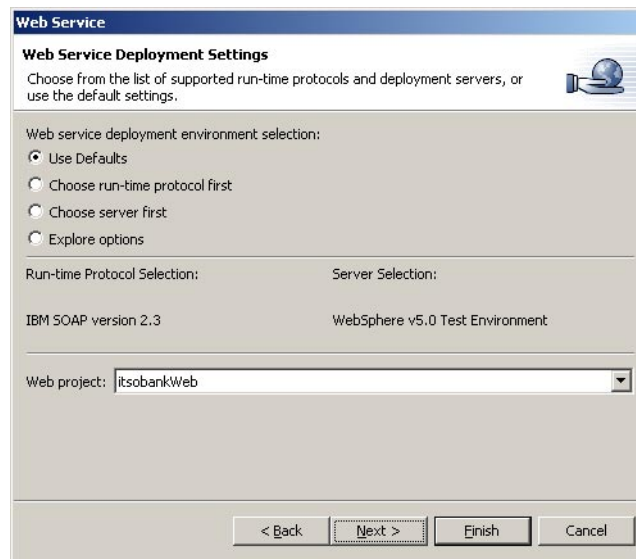
Figure 7-4   Web Service Deployment Settings

6. For Web Service Deployment Settings, select **Use Defaults** that uses Run-time protocol selection as IBM SOAP version 2.3 and Server selection as ITSOBank which we already creatred.
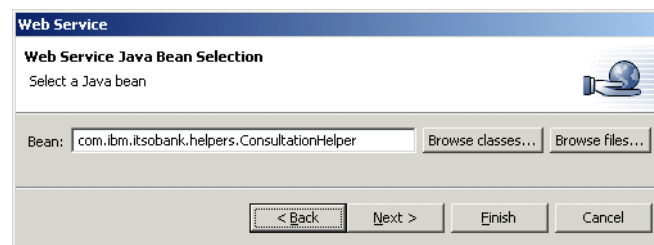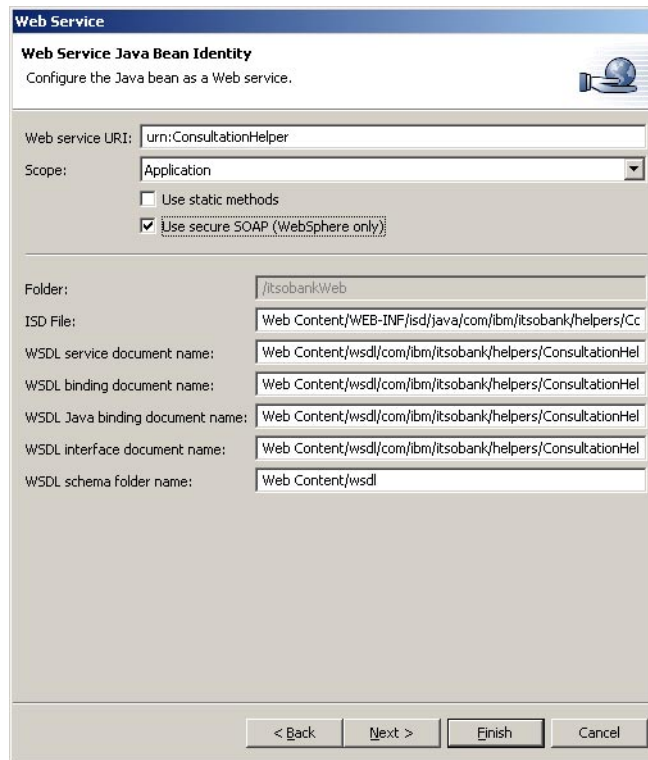
Figure 7-5   Web Service JavaBean selection

7. In Selecting JavaBean window the class name should come up as indicated. If not then click **Browse Classes** and select **ConsultationHelper**, then click **Next.**

Figure 7-6   Web Service Java Bean identity

8. In Web Service Java Bean Identity window, change WebServcice URI to
    `urn:ConsultationHelper` and make sure you check the **Use secure SOAP**
    check box as shown in Figure 8-6. Click **Next.**
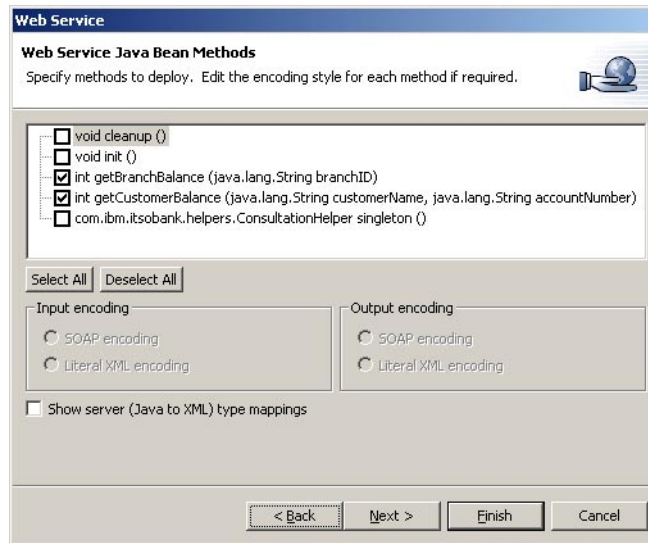
Figure 7-7   Web Service Java Bean Methods

9. On the Web Service Java Bean Methods window, select only **getBranchBalance()** and **getCustomerBalance()** methods. To check the type mappings you may like to click Show server (Java to XML) type mappings. Click **Next**.
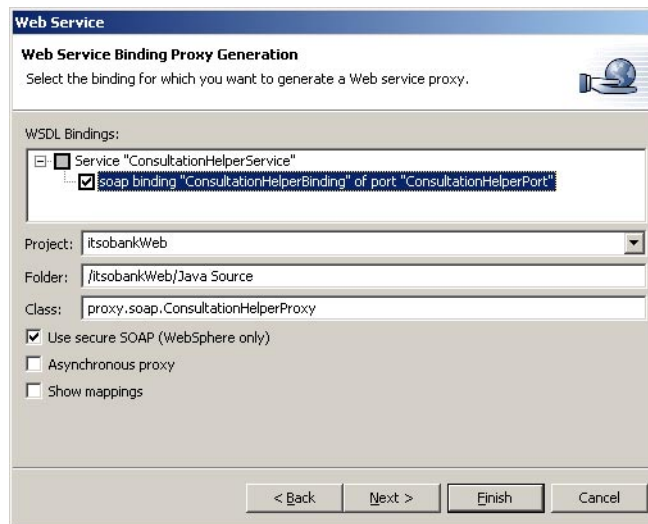
Figure 7-8   Web Service Binding Proxy Generation

10. On the Web Service Binding Proxy Generation window, change the Class to `com.ibm.itsobank.wsproxy.ConsultationHelperProxy`. Also make sure **Use secure SOAP(WebSphere Only)** is checked. Click **Next.**
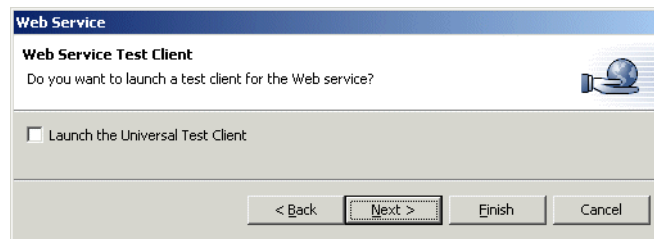
Figure 7-9   Web Service Test Client

11. In Web Service Test Client window, do not select Launch the Universal Test Client. click **Next.**
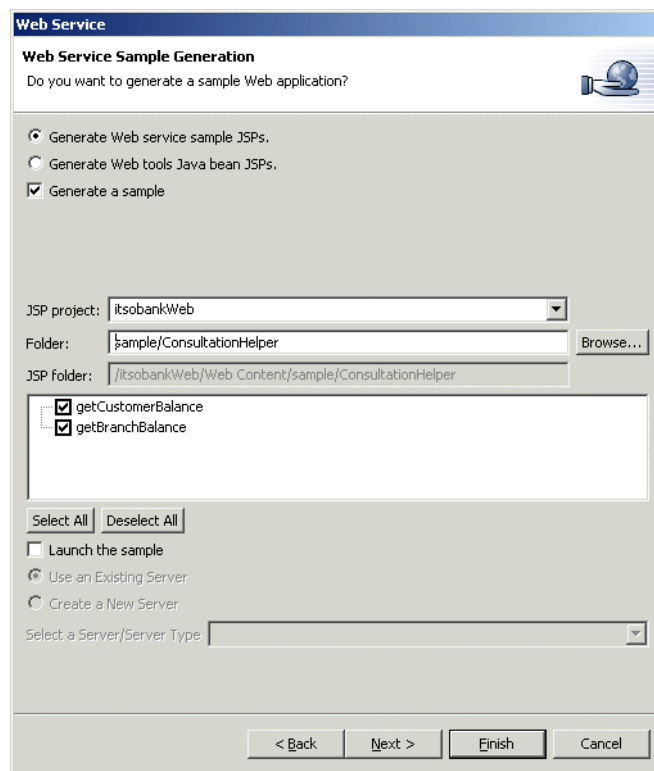
Figure 7-10   Web Service Sample Generation

12. As shown in the Figure 7-10 on page 133, in Web Service Sample Generation
    window, make sure Generate Web Services sample JSPs, Generate a
    sample and the two methods are selected. Click **Next.**
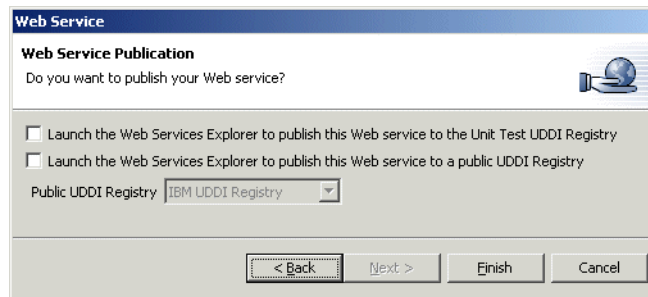
Figure 7-11   Web Service Publication

13. We reached the end of the Web services wizard. Since we will not publish this
    Web Service leave both the check boxes unselected. Click **Finish**.

After completing the last window of the wizard, it will take a couple of minutes for
WebSphere Studio to generate all the code depending on your machine's
performance. Once it has finished, the WebSphere Test Environment in
WebSphere Studio Application Developer will start automatically.

Looking at the generated code

After generating the code for the Web service, the following files will appear in
the project:

Table 7-1   Generated files for the new Web service

File name	Purpose
admin/*	Administering the Web services
conf/sig-config.dtd	DTD for the signature header handler
conf/ver-config.dtd	DTD for the verification header handler
conf/cl-editor-config.xml	Client's envelope editor descriptor
conf/cl-sig-config.xml	Client's signature header handler
conf/cl-ver-config.xml	Client's verification header handler
conf/sv-editor-config.xml	Server's envelope editor
conf/sv-sig-config.xml	Server's signature header handler

File name	Purpose
conf/sv-ver-config.xml	Server's verification header handler
key/SOAPclient	Client's keyring file in JKS
key/SOAPserver	Server's keyring file in JKS
key/sslserver.p12	Keyring file for the SSL connection in PKCS12 format
log/dummy.log	Dummy file in the log directory
sample/Consultation/*	Sample application for testing purposes
wsdl/Consultation-binding.wsdl	WSDL binding configuration file
wsdl/Consultation-service.wsdl	WSDL binding service file
dds.xml	Deployment descriptor for all the services
soap.xml	SOAP server configuration file
WEB-INF/classes/com/ibm/itsobank/wsproxy/ConsultationHelperProxy.class	SOAP Client for sample application
WEB-INF/isd/java/com/ibm/itsobank/wsproxy/Consultation.isd	Deployment descriptor for the particularservice
WEB-INF/lib/*	Java libraries required for Web services

Important: The generated code has two JKS key files, one for the SOAP server and one for the SOAP client. The keys provided are out of date, unfortunately they will not work. The keys provided with the ITSOBank sample are new, updated keys for testing purposes.

For your runtime environment you will want to use your own key files.

Testing the Web Service

After generating the code, you may want to test the code.

Note: The test client has been removed from the ITSOBank sample application, although you will find some of the code in one of the functions implemented using the Web Services.

Follow the steps below to use the generated test client in order to test the Web Service.

1. Open the Server perspective or switch to it if you have it already opened.

2. Find the following ConsultationHelper folder by clicking **itsobankWeb -> Web Content -> sample -> ConsultationHelper**.

3. Right-click the **TestClient.jsp** file, then select **Run on Server**. A Web browser opens with the requested URL.

   A window appears with three frames. In the leftmost frame, the following methods should be listed:

   – setEndPoint

   – getEndPoint

   – getBranchBalance

   – getCustomerBalance

4. Click the **getBranchBalance** link.

5. The upper-right frame changes: a text filed appears for you to input the branch ID. Type **raleigh** into the field, then click **invoke**.

6. After invocation the result frame changes and, after a couple of seconds and several messages on the console, the result should appear in the frame.

7. Check the getCustomerBalance method also, with the following parameters:

   – Customer ID: `johnd`

   – Account Number: `11223344-12345678`

   If the code is working, you should see the proper numbers under the result frame after invoking the methods with the parameters.

XML-SOAP Admin Tool

The Web services Admin tool is provided for the sample Web services and is generated automatically with the code. Start a browser, then enter the following URL:

```
http://<your_server>/itsobank/admin/index.html
```

Click the list all services link to see all the available services for this sample; you should see the picture shown in Figure 7-12.
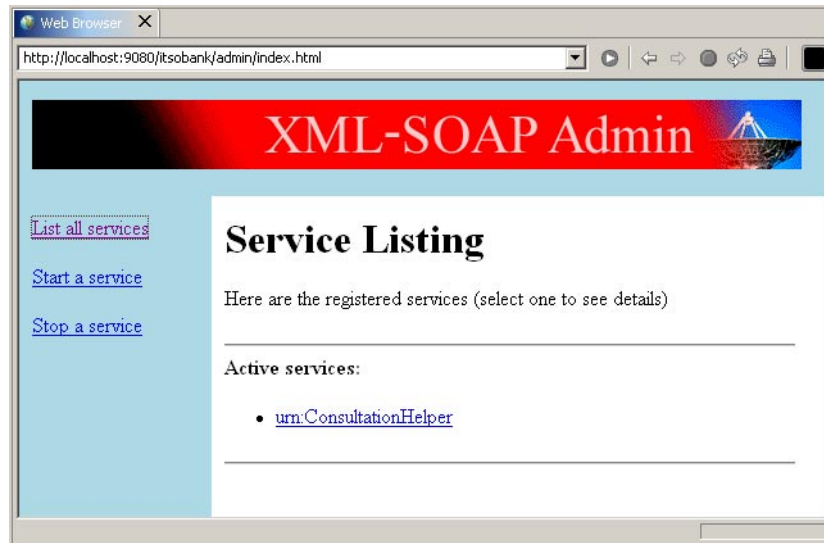
Figure 7-12   XML-SOAP Admin

The service listing shows one service available: *urn:ConsultationHelper*.

The services can be stopped or started under the related links within the XML-SOAP Admin tool.

Developing the ITSOBank Web service client

You will find a Web Service sample client developed based on the sample client code generated by the Web Services wizard.

The main code for the sample client came from the results.jsp file; most of the code was reused from there and modified to fit into the sample application. There are only two pages used for the Web Services client sample, find them under the Web module, *websvc* directory. The *query.html* file collects the input for the Web Service invocation on a HTML form; the *websvcresult.jsp* contains the code to invoke the Web service and present the results on the resulting page. For more information open the files and check the comments in the source.

If global security is enabled the Web service client will not work as it is, because the Web resources for the client (query.html, websvcresult.jsp, rpcrouter) are not secured, no user authentication is performed, no credentials are propageted while invoking the service. In order to resolve this problem, either you can disable global security for testing purposes or you can secure the Web resources listed previously by following the same process from Section 7.1.2, "HTTP Basic Authentication" on page 143.

Using the ITSOBank Web service client

You can access the ITSOBank Web service client from the index page, by selecting the **Consultation Client** link. On the next page set the endpoint in case you need to, by default the Web service is hosted on the same system where the client is running.

You can get the balance either for a customer account or a branch account, by filling out the input fields and submitting your request.

The returning result page should show the account balance of your choice.

Tracing SOAP Requests / Checking the logs

For better understanding, let us check SOAP request and SOAP response by monitoring TCP/IP packets that are exchanged between the server and the client.

1. Go to the **Server** Perspective.

2. Create a new server and a server configuration as depicted below.
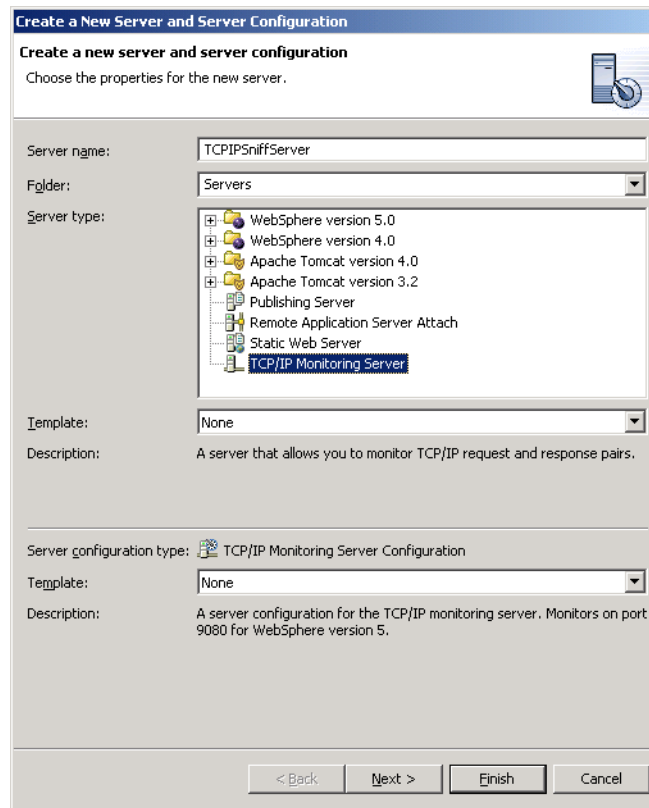
Figure 7-13   TCP/IP Monitoring Server

3. Click **Finish**.

4. Open Server configuration for the newly created TCP/IP monitoring server.

5. Set Local Port and Remote Port. For us, Remote Port will be **9080** and Local Port is **8080**. Local Port is the port TCP/IP Monitoring Server will work on and Remote Port is the one on which the actual server is running on.

6. Save this configuration and start the TCP/IP Monitoring Server.

Make changes of the port number for the test client of the Web service pointing to TCP/IP Monitoring Server instead of the actual server. Using **setEndPoint** method change the *rpcrouter* URL also to TCP/IP Monitoring Server URL. When you test getBranchBalance method, you will see the SOAP request as below.

Example 7-1   SOAP Request with certificate

```
POST /itsobank/servlet/rpcrouter HTTP/1.0
Host: localhost:9080
```

```
Content-Type: text/xml; charset=utf-8
Content-Length: 3933
SOAPAction: ""
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Header>
<SOAP-SEC:Signature SOAP-ENV:actor=""
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:mustUnderstand="1"
xmlns:SOAP-SEC="http://schemas.xmlsoap.org/soap/security/2000-12"><Signature
xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo Id="sig">
    <CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <Reference URI="">
      <Transforms>
        <Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>P5yoY11dxamU3aeMkYHqNm/XstE=</DigestValue>
    </Reference>
    <Reference Type="http://www.w3.org/2000/09/xmldsig#SignatureProperty"
URI="#timestamp">
      <Transforms>
        <Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>/vhLHTEeNcI7O58053YIFvTQwmw=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>
    WNYOBJzuDlAcMAh+DgHENluKRLt4CqcvqaQ4FMDbctRjrX1afovzg9SKJ1wfA/rBm8jKJ9hI
    nvhbeqkZqmMcaRl7ztWFpMPjXxRT8BiYl71pLAERavIaFmwddWteYhMBgEXOYkR2w1cGjaPq
    VAZJxGSn/SkvrmL3x2adXThXMmw=
  </SignatureValue>
  <KeyInfo>
    <KeyName>soaprequester</KeyName>
    <KeyValue>
      <RSAKeyValue>
        <Modulus>
          zLc99nWY+GsSwG9iI64iU9XdSKzljLqGbGjZjBgLacOME/MqyVZSL9D58r4M11jooQ
          ea4OtxVwSZrjwn1fVrlq9GWSxB8qmOqhWfh4HKzaL/CTDLNCENoWOLv38y+dNFbwRX
          L3OUOrDg8WeE6h7W2UNwOG4gf98i4Y7POSVNX58=
        </Modulus>
```

```
          <Exponent>AQAB</Exponent>
        </RSAKeyValue>
      </KeyValue>
      <X509Data>
        <X509Certificate>
MIIDQTCCAqqgAwIBAgICAQQwDQYJKoZIhvcNAQEFBQAwTjELMAkGA1UEBhMCSlAxETAPBgNVBAgT
...
xW97LLNegQCO/b+aFD8XKw2U5ZtwbnFTRgsO97dmzO9RosDKkLlM
        </X509Certificate>
      </X509Data>
    </KeyInfo>
    <Object>
      <SignatureProperties>
        <SignatureProperty SOAP-SEC:id="timestamp" Target="#sig">
          <timestamp>Mon Jul 15 10:56:05 EDT 2002</timestamp>
        </SignatureProperty>
      </SignatureProperties>
    </Object>
</Signature></SOAP-SEC:Signature>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<ns1:getBranchBalance
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns1="urn:ConsultationHelper">
<branchID xsi:type="xsd:string"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">raleigh</branchID>
</ns1:getBranchBalance>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The SOAP Response will be something similar as the following example.

Example 7-2   SOAP Response

```
HTTP/1.1 200 OK
Server: WebSphere Application Server/5.0
Set-Cookie: JSESSIONID=0000WWFO2LRYHHRFWJHFO4DEJYQ:-1;Path=/
Cache-Control: no-cache="set-cookie,set-cookie2"
Expires: Thu, 01 Dec 1994 16:00:00 GMT
Content-Type: text/xml; charset=utf-8
Content-Length: 3743
Content-Language: en-US
Connection: close
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Header>
```

```
<SOAP-SEC:Signature
xmlns:SOAP-SEC="http://schemas.xmlsoap.org/soap/security/2000-12"
SOAP-ENV:actor="" SOAP-ENV:mustUnderstand="1"><Signature
xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo Id="sig">
    <CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"></CanonicalizationMe
thod>
    <SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"></SignatureMethod>
    <Reference URI="">
      <Transforms>
        <Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"></Transform>
      </Transforms>
      <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
      <DigestValue>oVD/QxRYXFgO2v6rK53DZKntqlI=</DigestValue>
    </Reference>
    <Reference Type="http://www.w3.org/2000/09/xmldsig#SignatureProperty"
URI="#timestamp">
      <Transforms>
        <Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"></Transform>
      </Transforms>
      <DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
      <DigestValue>/vhLHTEeNcI7058053YIFvTQwmw=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>
    MpazDPPD1QIk7n5ledadOYC85DXnFIZHD9/jquVQdkj3q35yMLfflZLLtliO7byRt+WQpyEO
    JvIF7jHG1tN9Vueh2FANmhd5y5CI68uZHouXqEBU8rXIRJBOmI4xK5ZUYaONzL6rsNXbEvHr
    7hQcz9nkAtc65JC8Hak+yLhQqfO=
  </SignatureValue>
  <KeyInfo>
    <KeyName>soapprovider</KeyName>
    <KeyValue>
      <RSAKeyValue>
        <Modulus>
          raakNJ1JzkPUuvPdXRvPOOCl12nBwmqvt65dk/x+QzxxarDNwH+eWRbLyyKcrAydOX
          GV+Zbvj6V3O9DSVCZUCJttw6bbqqeYhwAP3V8s24sID77tk3gOhUTEGYxsljX2orL2
          6SLqFJMrvnvk2FRS2mrdkZEBUG97mD4QWcln4dO=
        </Modulus>
        <Exponent>AQAB</Exponent>
      </RSAKeyValue>
    </KeyValue>
    <X509Data>
      <X509Certificate>
```

```
MIIDQDCCAqmgAwIBAgICAQUwDQYJKoZIhvcNAQEFBQAwTjELMAkGA1UEBhMCSlAxETAPBgNVBAgT
...
Ss47F4D6woPsAd2ubg/YhMaXLTSyGxPdV3VqQsutuSgDUDoqWCA=
      </X509Certificate>
    </X509Data>
  </KeyInfo>
  <Object>
    <SignatureProperties>
      <SignatureProperty Target="#sig" SOAP-SEC:id="timestamp">
        <timestamp>Mon Jul 15 10:56:05 EDT 2002</timestamp>
      </SignatureProperty>
    </SignatureProperties>
  </Object>
</Signature></SOAP-SEC:Signature>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<ns1:getBranchBalanceResponse xmlns:ns1="urn:ConsultationHelper"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<return xsi:type="xsd:int">99800</return>
</ns1:getBranchBalanceResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Another way to look at these details is through checking the log files. Under
itsobankWeb project -> Web Content -> log the *SOAPVHH-all-sv.log* file
contains all the log entries of the SOAP requests with certificates that comes to
the server and the SOAPVHH-all-cl.log contains the SOAP responses.
SOAPVHH-fail-cl.log and SOAPVHH-fail-sv.log contains all the failed requests
and responses.

7.1.2 HTTP Basic Authentication

The following sections will explain how to enable security in WebSphere
Application Server V5 using the Administrative Console.

HTTP Basic Authentication

The following steps show how to secure the rpcrouter servlet in the enterprise
application and use Basic Authentication for authentication purposes.

1. In WebSphere Studio go to the **Server** perspective.

2. In the Server Configuration expand the **Server Configurations** double-click
   the ITSOBank server. This opens the server configuration window.

3. In that window, under **Security -> Cell Settings** select the **Enable Security**
   check box.

4. Below that check box, for **Local OS Authentication** provide the details of **Server ID**, **Server Password** and **Confirmed Password**.

5. Save this configuration.

6. Open **web.xml** (Web Deployment Descriptor) under the **itsobankWeb -> Web Content -> WEB-INF** folder.

7. Click **pages** tab of the deployment descriptor view.

8. In this view, under **Login** in the drop down list for **Authentication method** select **Basic**. You can give a name to this Realm in the text box provided for Realm name.

9. In the same window, switch to the **Security** tab.

10. **Add** a new **Security Role**, for example: `webservicesrole`.

11. And click **Security Constraints** button just below the Securiy heading.

12. Now **Add** a **Security Constraints**. Now under **Web Resource Collections** you should see an entry named **(New Web Resource Collection)**. Select that and click **Edit**. This opens a new window called *Web Resource Collections*.

13. Under HTTP Methods select **GET** and **POST**. We need to protect the URLs by HTTP Basic Authentication. First let us protect the **rpcrouter** servlet. For that add **/servlet/rpcrouter** to the URL section.

14. Save the Web deployment descriptor.

15. Under **itsobank -> META-INF** open application.xml.

16. Switch to the **security** tab, click **Gather**, this will gather all the security roles defined for all the modules.

17. Now click the security role that we have defined, webservicesrole, then under WebSphere Bindings select **All authenticated users**.

18. Save and close the configuration.

Go to server perspective to re-start the server. This is done so that the server picks up the security information that we have defined.

We also need to modify the code that is generated by WebSphere Studio to make this HTTP Basic Authentication work for us. For this create a new proxy client without WebSphere Security for better understanding. Please refer to the redbook *Web Services Wizardy with WebSphere Studio Application Developer SG24-6292* for more information.

Modify the Web services proxy code generated by the wizard to put basic authentication credentials.

Open the Web Service proxy client that is generated by WebSphere Studio Application Developer. In this class, you will see the methods that we exposed as Web services, for example in our case you would see *getBranchBalance* and *getCustomerBalance* methods, add the following lines to the code.

Example 7-3   Setting user name and password for the SOAP transport

```
SOAPHTTPConnection soaptransport = new SOAPHTTPConnection();
soaptransport.setUserName("your-userName");
soaptransport.setPassword("your-Password");
call.setSOAPTransport(soaptransport);
```

Save the proxy class and re-start the test environment server. Now when you test your Web Service again, if you use TCP/IP Server to chech the SOAP request, the SOAP Header will consist of user credentials. But this information is by default encrypted using Base64 encryption algorithm. So you won't be able to read the user name and password.

HTTP Basic Authentication with SSL

It is very easy to use these Web Services, we developed, using SSL. Make sure you have HTTPS port enabled for your test environment in WebSphere Studio by checking it on server configuration page.

We know the URL for our test Web service sample as
`http://localhost:9080/itsobank/sample/ConsultationHelper/TestClient.jsp`
. For using SSL we just need to change the port number from 9080 to 9443 in the above URL and check
`http://localhost:9443/itsobank/sample/ConsultationHelper/TestClient.jsp`
in the browser, you should see the certificate appear.

Secured and non-secured services together

It is possible many times that we need to secure some services and we do not need to secure some services. Even for the services we secure, if all those services are accessed using the same URL then if a user supplies the credentials for one service, with that the user can access any service. To protect this, we need to create different URLs for each secured service and allow them for different users, groups and roles.

For example create two servlet URLs, one is protected and one is not. If you see the web.xml sample below, you will find that this is very simple, as you just have to create another URL for services. And this security configuration is defined in the above sections.

Example 7-4   Secured and non-secured Web Services together

```
<servlet>
```

```
     <servlet-name>unprotectedRPCRouter</servlet-name>
     <display-name>unprotectedRPCRouter</display-name>
     <servlet-class>com.ibm.soap.server.http.WASRPCRouterServlet</servlet-class>
     <init-param>
        <param-name>FaultListener</param-name>
        <param-value>org.apache.soap.server.DOMFaultListener</param-value>
     </init-param>
</servlet>
<servlet>
     <servlet-name>protectedRPCRouter</servlet-name>
     <display-name>Apache-SOAP RPC Router</display-name>
     <description>no description</description>
     <servlet-class>com.ibm.soap.server.http.WASRPCRouterServlet</servlet-class>
     <init-param>
        <param-name>faultListener</param-name>
        <param-value>org.apache.soap.server.DOMFaultListener</param-value>
     </init-param>
</servlet>
```

This security configuration can even be done using Application Assembly Tool(AAT). For more information you can look at Chapter 10, "Administering WebSphere security" on page 233

7.1.3 WS-Security

The Web Services Security specification (WS-Security) provides a set of mechanisms to help developers of Web Services secure SOAP message exchanges. Specifically,WS-Security describes enhancements to the existing SOAP messaging to provide *quality of protection* through the application of message integrity, message confidentiality, and single message authentication to SOAP messages. Additionally, WS-Security describes how to encode binary security tokens(security token represents a collection of claims like name, identity, key, group, privilege, capability and so on) and attach them to SOAP messages.

Security tokens assert claims which can be coupled with digital signatures to provide mechanisms for demonstrating evidence of the sender's knowledge of the keys described by the security token. In addition the definition of a SOAP header element provides a mechanism for "binding" or "associating" the signature with the claims in the security token.

WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. These mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

WS-Security also provides a general-purpose mechanism for associating security tokens with messages. No specific type of security token is required by WS-Security. It is designed to be extensible, for example: support multiple security token formats, or a client might provide proof of identity and proof that they have a particular business certification.

Additionally, WS-Security describes how to encode binary security tokens. The specification describes how to encode X.509 certificates and Kerberos tickets as well as how to include opaque encrypted keys. It also includes extensibility mechanisms that can be used to further describe the characteristics of the credentials that are included with a message.

Web Services security specifications

WS-Security only provides foundation for other security specifications. Layered on this we have a policy layer model (WS-Policy), a trust model (WS-Trust), and a privacy model (WS-Privacy). These specifications provide us the foundation to establish secure interoperable Web Services across the domain. The follow-on specifications include secure conversation (WS-SecureConversation), federated trust (WS-Federation), and authorization (WS-Authorization). All these specifications should provide security framework specifications related to auditing, management, and privacy.

Follow-On Specifications	WS-SecureConversation	WS-Federation	WS-Authorization
Initial Specifications	WS-Policy	WS-Trust	WS-Privacy
WS-Security	WS-Security		
SOAP Foundation	SOAP Foundation		
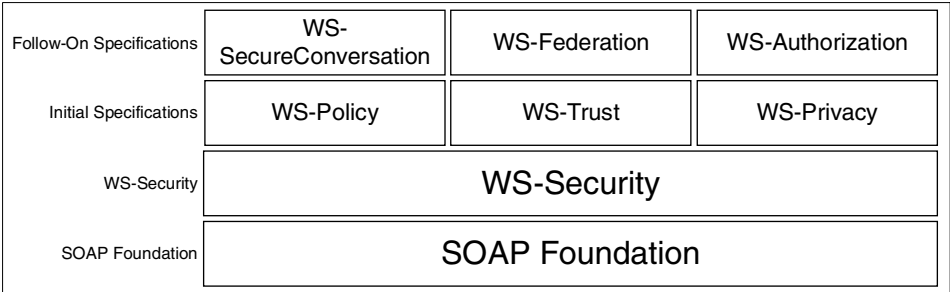
Figure 7-14   Web Services Security Specifications

The following sections will provide more details on the initial specifications.

WS-Policy

WS-Policy describes the capabilities and constraints of the security policies on intermediaries and endpoints. This way senders and receivers can define their requirements and capabilities.

WS-Policy will be fully extensible and will not place limits on requirements and capabilities that may be described. However the specification will likely identify several basic service attributes, encoding formats, security token requirements and supporting algorithms. This specification will define a generic SOAP policy format, which can support more than just security policies. This specification will also define a mechanism for attaching service policies to SOAP messages.

WS-Trust

WS-Trust describes a framework for trust models that enables Web Services to securely interoperate. This establishes the model for both direct and brokered trust relationships.

WS-Trust specification will describe how existing direct trust relationships may be used as the basis for brokering trust through the creation of security token issuance services. These security token issuance services build on WS-Security to transfer the requisite security tokens in a manner that ensures the integrity and confidentiality of those tokens. This specification will also describe how several existing trust mechanisms may be used in conjunction with this trust model.

Finally, the trust model will explicitly allows for, but will not mandate, delegation and impersonation by principals. Note that delegation is consistent with impersonation, but provides additional levels of traceability.

WS-Privacy

WS-Privacy describes a model for how Web Services and requesters state subject privacy preferences and organizational privacy practice statements.

By using a combination of WS-Policy, WS-Security, and WS-Trust, organizations can state and indicate conformance to stated privacy policies. This specification will describe a model for how a privacy language may be embedded into WS-Policy descriptions and into WS-Security.

Web Services security model

SOAP message acts as a requester to the Web Service and the response from the Web Service is also a SOAP message. So protecting the message content from illegally accessed (confidentiality) or illegally modified (integrity) are the primary security concerns of Web Services.

Today's Web service application topologies include a broad combination of mobile devices, gateways, proxies, load balancers, demilitarized zones (DMZs), outsourced data centers, and globally distributed, dynamically configured systems. All of these systems rely on the ability for message processing

intermediaries to forward messages. Specifically, the SOAP message model operates on logical endpoints that abstract the physical network and application infrastructure and therefore frequently incorporates a multi-hop topology with intermediate actors.

Point-to-Point Configuration

The available Security solutions like Secured Socket Layer (SSL) / Transport Layer Security (TLS) and IPSec like network layer solutions provide features like authentication, data integrity and data confidentiality. But the main problem is that these solutions enable only point-to-point secure sessions.
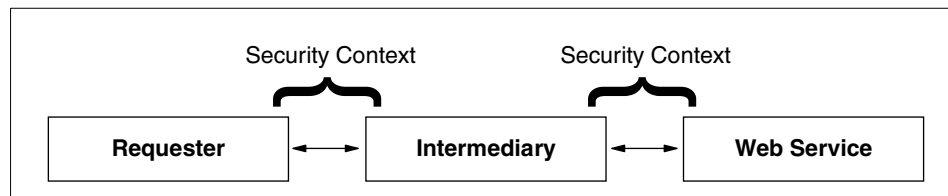
Figure 7-15   Point-to-Point Configuration

End-to-End Configuration

When data is received and forwarded on by an intermediary beyond the transport layer both the integrity of data and any security information that flows with it maybe lost. This forces any upstream message processors to rely on the security evaluations made by previous intermediaries and to completely trust their handling of the content of messages. What is needed in a comprehensive Web service security architecture is a mechanism that provides end-to-end security. Successful Web service security solutions will be able to leverage both transport and application layer security mechanisms to provide a comprehensive suite of security capabilities.
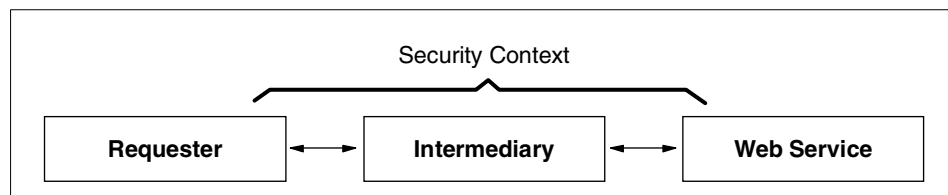
Figure 7-16   End-to-End Configuration

Security Token Service Model

Following explains the Security Token Service Model to enable us to achieve end-to-end security goals. Figure 7-17 explains us that any requester may also be a service, and that the Security Token Service may also fully be a Web service, including expressing policy and requiring security tokens.
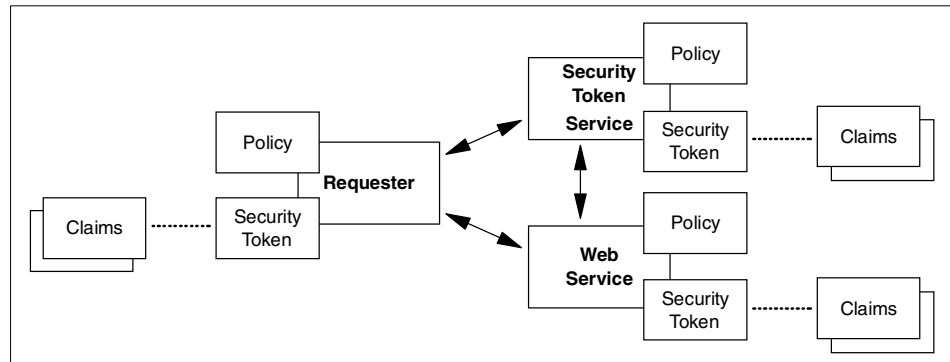
Figure 7-17   Security Token Service Model

A Web service can require that an incoming message prove a set of claims (for example: name, key, permission, capability, and so on). If a message arrives without having the required claims, the service may ignore or reject the message. We refer to the set of required claims and related information as policy

A requester can send messages with proof of the required claims by associating security tokens with the messages. Thus, messages both demand a specific action and prove that their sender has the claim to demand the action.

When a requester does not have the required claims, the requester or someone on its behalf can try to obtain the necessary claims by contacting other Web services. These other Web services, which we refer to as security token services, may in turn require their own set of claims. Security token services broker trust between different trust domains by issuing security tokens.

Scenarios

The following sections discuss some of the WS-Security scenarios. There are only a few of the possible scearions are covered here, those that will give you an introduction to the topic and an easy understandig of the topic.

Direct Trust using basic authentication and Transport-Level Security

In this scenario the requester opens a connection to the Web Service by exchanging a public key pair to establish a secure channel over an HTTP connection. Then the server prompts for user ID and password through an HTTP message exchange and these user credentials are carried through the HTTP headers.
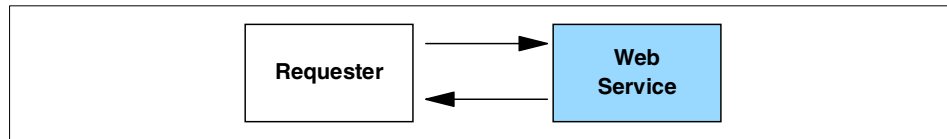
Figure 7-18   Direct Trust using basic authentication and Transport-Level Security

The client opens a connection to the Web Service using secure transport. It sends its request and includes a security token that contains its username and password. The service authenticates the information, processes the request and returns the result.
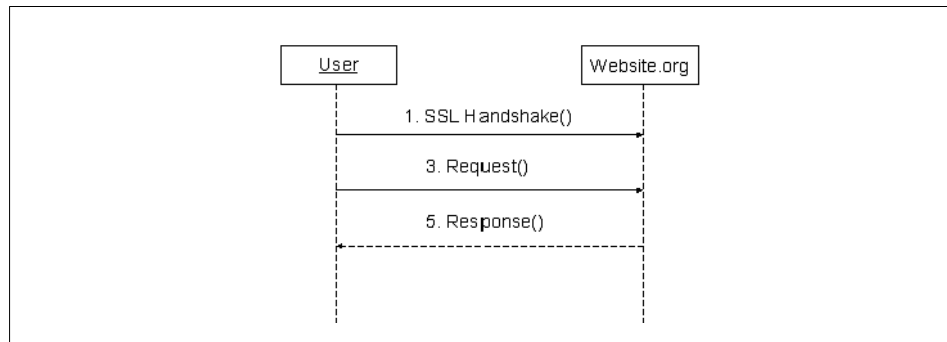
Figure 7-19   Sequence of events for Scenario using <UsernameToken>

In this scenario, the message confidentiality and integrity are handled using existing transport security mechanisms.

Figure 7-19 shows the sequence of events for this scenario.

1. The client opens a connection to the Web Service using a secure transport such as SSL.

2. The client constructs a SOAP message. There is a <UsernameToken> element in the <Security> header, this element contains the client's username and password for the service. The password can be sent in as plain text because the transport layer is secure.

3. The message is sent to the service.

4. The service extracts the <UsernameToken> element and validates the user name and password.

5. Since the validation succeeded, the service processes the request and returns the result.

Direct Trust using Security Tokens

This scenario illustrates the use of a security token that is directly trusted by a Web service.
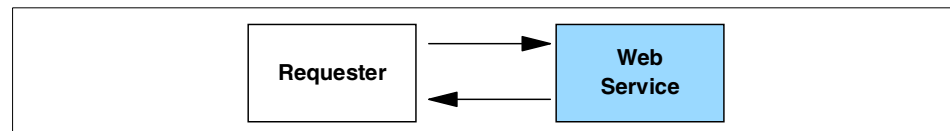
Figure 7-20   Direct Trust using Security Tokens

Here direct trust means that the requester's security token (or its signing authority) is known and trusted by the Web service. This scenario assumes that the two parties have used some mechanism to establish a trust relationship for use of the security token. This trust may be established manually, by configuring the application, or by using a secure transport to exchange keys. By secure transport of keys we mean that a transport such as SSL (or other mechanism or process) can be used as a way for a trusted party to assert the validity of a key or security token to a recipient party. No assumption is made about the organizational relationship between the parties.
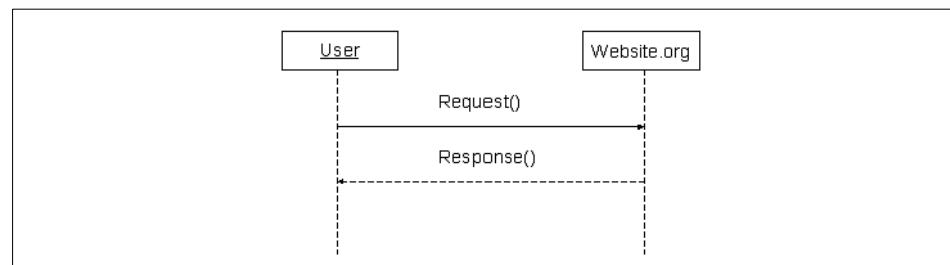
Figure 7-21   Sequence of events for using Direct Trust using Security Tokens

Figure Figure 7-21 shows the sequence of events for this scenario. As you can see there is no SSL handshake as it happened in the previous scenario.

1. The client sends a message to a service and includes a signed security token and provides proof-of-possession of the security token.

2. The service verifies the proof and evaluates the security token.

3. If the signature on the security token is valid and is directly trusted by the service then it processes the request and returns a result.

Security Token Acquisition

In some cases, the security token used is not passed as part of the message. Instead, a security token reference is provided that can be used to locate and acquire the token.
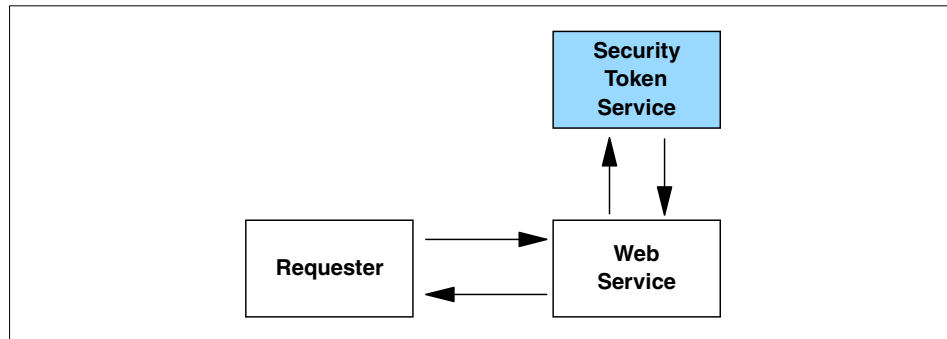
Figure 7-22   Security Token Acquisition

Figure 7-23 shows the main events in the scenario of using Security Token Acquisition.
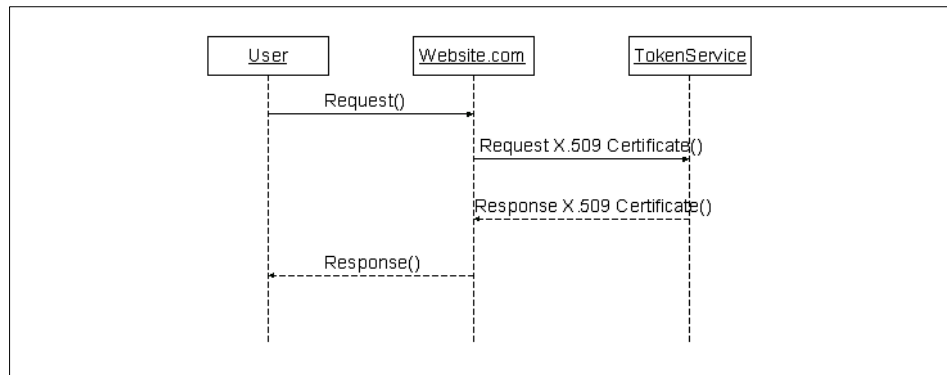
Figure 7-23   Scenario using Security Token Acquisition

Following are the series of events that happens in this scenario.

1. The client constructs the SOAP message.

2. The client computes a signature for the key elements of the message using XML Signature.

3. The client's X.509 certificate is at a specific Web location. Consequently, rather than pass it in the message, the client provides a reference to its X.509 certificate using the <SecurityTokenReference> element in the <Security> header.

4. The client sends the request to the service.

5. The service extracts the reference to X.509 certificate from <Security> header.

6. The service fetches the certificate from the specified URL location.

7. The service verifies the certificate, its validity and its signature.

8. When the service authenticates the certificate it processes the message and returns a result.

Firewall Processing

Firewalls remain a critical component of the Web Services security architecture and WS-Security Specifications also addresses security on Firewall.
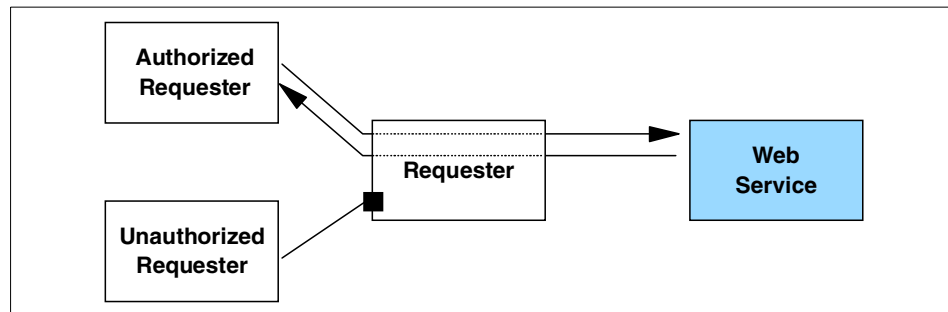
Figure 7-24   Firewall processing

As shown in the Figure Figure 7-25 firewall processes the incoming SOAP messages and only allows those from authorized clients to penetrate the firewall.
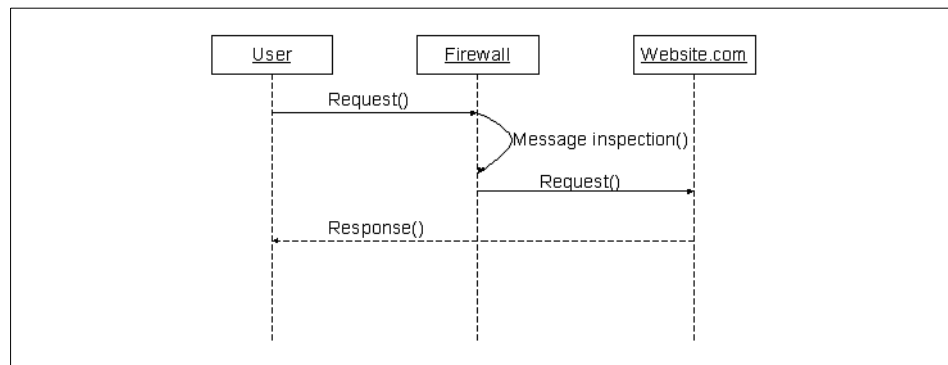
Figure 7-25   Scenario using Firewall Processing

In this scenario, firewall observes the security tokens used to sign the message. If the signature is valid, and the signing the authority for the security token is trusted to authorize messages into the firewall, and then message is allowed; otherwise it is rejected. In some cases, a signature may specifically reference the firewall as a SOAP actor.

This scenario would operate as follows:

1. The client constructs a SOAP message and sends it to a Web Service.

2. The message passes through a firewall prior to arriving to the Web Service.

3. The firewall examines the security token(s) and signatures in the <Security> header.

4. Firewall makes an assessment on the validity of the message and possibly using external data, makes a determination as to whether or not to authorize the message passing through the firewall.

When the SOAP message is without any encryption the firewall will be able to examine the message headers for authorization, but what if it is encrypted? As an end-to-end security SOAP message is encrypted. However it can still validate message as follows:

1. After signing and encrypting the message, the client adds an additional <Security> header with the firewall listed as the SOAP actor. Into this header block the client places a <ds:Signature> element containing a signature over the encrypted data. Also using <BinarySecurityToken> the signature is prepended.

2. From <Security> header firewall reads and validated security token and signature.

3. The firewall then makes a determination, possibly using external data, as to whether or not to authorize the message to pass through the firewall.

7.1.4 Security with the Web Services Gateway

WebSphere Web Services GateWay is bundled with WebSphere Application Server V5 Network Deployment package. After installing Network Deployment, we need to install the wsgw.ear and wsgwsoap1.ear on the application server.

Gateway Security Implementation

Web Services GateWay provides HTTP Basic Authentication and authorization mechanism based upon the security features provided by WebSphere Application Server.

> **Important:** Gateway-level authentication must be enabled for enabling operational-level authentication. You must do the same even before installing channels. When we enable gateway-level authentication, filters will have access to the requesto's authentication information.

Security can be applied at two levels:

▶ Enable gateway-level authentication

▶ Enable operation-level authorization

Enable gateway-level authentication

Providing gateway-level authentication is actually nothing but protecting the URLs */axisengine* and */soaprpcrouter* with HTTP Basic Authentication. And these URLs are of the channel application *wsgwsoap1.ear* and *wsgwsoap2.ear*. We need to modify the web.xml files of these EAR files. Extract EAR files, again extract wsgwsoap1.war, wsgwsoap2.war files and finally extract web.xml files to modify.

Example 7-5   BASIC security enabling for wsgwsoap1.ear and wsgwsoap2.ear

```
<!-- Define a Security Constraint on this Application -->
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>SOAP Entry Servlets</web-resource-name>
      <url-pattern>/axisengine</url-pattern>
      <url-pattern>/soaprpcrouter</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <!-- Anyone with one of the listed roles may access this area -->
      <role-name>your role name - for example "meterable"</role-name>
    </auth-constraint>
  </security-constraint>
  <!-- Define the Login Configuration for this Application -->
  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>your realm name - for example "WSGW Metered Access
Area"</realm-name>
  </login-config>
  <!-- Define the Security Role to use -->
  <security-role>
    <description>your security role - for example "WSGW meterable
role"</description>
    <role-name>your role name</role-name>
  </security-role>
```

Make sure to enable **Global Security** for HTTP Basic Authentication in WebSphere Application Server using Admin Console as mentioned in "HTTP Basic Authentication" on page 143

> **Note:** Make a copy of the wsgwsoap1.ear and wsgwsoap2.ear before modifying them for enebling gateway-level authentication.

Enable operation-level authorization

First of all, make sure you have enabled Gateway-level authentication before enabling operation-level authorisation. The operation-level authorization is only possible for those Web Services that are deployed onto Web Services GateWay (WSGW) with the option **Authorization Policy-Control access to this service** selected.

Providing Operation-level authorization is also nothing but extending the WebSphere security for the beans to Web services. That is the reason why operation-level authentication is also called as *Web Service Security - Role-based authorization*.

The implementation consists of writing a wrapper enterprise bean for the actual Web service component, making sure that the method calls are matching. Now implementing WebSphere Application Server's authorization methodology into the wrapper enterprise bean will make sure that the Web service component is invoked only when the caller has appropriate authorization levels. We need to protect the service component and the enterprise bean in the .ear file and apply authorization rules to the EAR file.

Following procedure will show you how to implement operation-level authorization:

1. Create EAR File:

    a. Go to <WSGW_root>/scripts/auth folder in the command prompt.

    b. Run the following command:

    `WSGWAuthGen <location> <your_service>`

    Where the <location> is the URL for the gateway and <your-service> is the Name of the service as deployed onto the gateway (Case-sensitive).

    For example:

    `WSGWAuthGen http://myserver:port_number/wsgw Calendar`

    c. The `your_service.ear` file should now be created in the <WSGW_root>/scripts folder.

2. Assign roles to protect methods:

    a. In WebSphere Application Assembly Tool (AAT), select **File -> Open** and find the file wsgwauth.ear file under <WebSphere_ND_root>/installableApps.

    b. Following steps will import the `your_service.ear` archive into the wsgwauth.ear archive.

        i. On the navigation pane, open the pop-up menu for **EJBModules** and select **Import**.

    ii. Browse to select as file <WSGW_root>/scripts/your_service.ear. The Select modules to import window opens.

    iii. Select **your_service.ear** and click **OK**.

    iv. Click **OK** for confirmation.

    v. On the navigation panel, expand EJB Modules to make sure that your_service.ear has been imported.

c. Expand **EJBModules -> Your_service.ear** and select **Security Roles**.

d. For every security role that we need to create, we need to perform the following steps repeatedly:

    i. Select **New** from the pop-up menu for Security Roles.

    ii. Type name and description of the new security role and click **OK**.

e. Expand **EJBModules -> Your_service.ear** and select **Method Permissions**.

f. For every defined role that we need to a Web Service method, we need to perform the following steps repeatedly:

    i. Select **New** from the pop-up menu for Method Permissions.

    ii. Type the name of the new method permission and click **Add** for methods.

    iii. In the Add methods window, expand the tree for remote methods and select the method to be protected. Click **OK**, then the Add Methods window closes.

    iv. In the New Method Permission window, click **Add** for Roles. Select a previously defined role from the list then click **OK**.

g. Set the EJBReferences to ensure that the authorization enterprise bean can refer the newly imported enterprise bean. Complete the following steps:

    i. In the navigation pane, expand **WSGW Authorization group -> Session Beans -> Authorization** and select **EJB References**.

    ii. To open a new EJB Reference window, select **New** from the pop-up menu for EJB References.

    iii. In the New EJB Reference window, under the General tab, type a name for the reference then use the Link combination box to select the newly-imported EJB (all the other fields on this tab are populated automatically).

    iv. In the New EJB Reference window, under the Bindings tab, type the JNDI name as it appears in the bindings tab of the service enterprise

bean, this should be in the form
`websphere/WSGW/Security/your_service`.

    v. Click **OK**.

  h. From Application Assembly Tool file menu, select **File -> Save** to save the file wsgwauth.ear.

  i. Close Application Assembly Tool.

3. Install updated wsgwauth.ear:

  a. Start Administrative Console of WebSphere Application Server.

  b. On the navigation pane, select **Applications -> Install an Application**.

  c. Use **Install New Application** to install wsgwauth.ear.

  d. Go through the deployment process for the enterprise application. Select the users or groups to be assigned to the roles when prompted.

  e. At the end, **Finish** the deployment and save the configuration for WebSphere.

7.2 Messaging security

This chapter explains how to provide messaging security for the WebSphere Application Server. WebSphere Application Server V5 ships with an Embedded JMS Provider which simplifies messaging for the application server. At the same time WebSphere Application Server provides the flexibility to use any other JMS Providers like WebSphere MQ or JMS providers from other vendors.

This chapter provides useful information about how to configure security when using Embedded JMS Provider and even WebSphere MQ as the JMS Provider.

7.2.1 Messaging security

Java Messaging Service (JMS) is a Java API that allows applications to create, send, receive, and read messages. JMS API in the J2EE platform has the following features.

► Application clients, Enterprise JavaBean components and Web components can send or synchronously receive a JMS message. Application clients can in addition receive JMS messages asynchronously.

► A new kind of enterprise bean, the message-driven bean, enables the asynchronous consumption of messages. A JMS provider may optionally implement concurrent processing of messages by message-driven beans.

► Message sends and receives can participate in distributed transactions.

The JMS specifications does not discuss the security and encryption of the message that is getting transfered using the JMS provider. Instead specifications leave the security implementation to the JMS provider. Here in this section later we are going to discuss WebSphere MQ as a JMS provider.

Security Services

This section will investigate the five security services for messaging.

► **Authentication** is a mechanism to check whether the application or the user is genuine or not. In WebSphere MQ context, when a message channel starts, it is possible for the message channel agent (MCA) at each end of the channel to authenticate its partner, known as *mutual authentication*. For the sending MCA, this provides assurance that the partner it is about to send messages to is genuine. And for the receiving MCA there is a similar assurance that it is about to receive messages from a genuine partner.

   The application that handles the massaging has to do the authentication; for example: when a servlet sends a message WebSphere has to authenticate the user if he/she can run the servlet. Since there is no message level security (who can send what type of message) message level should be considered during application design.

► **Authorization** for the WebSphere MQ objects are stored in MQ (actually in a special queue). WebSphere MQ uses normal operating system user name and group authorizations to protect WebSphere MQ applications and WebSphere MQ Administration.

   Access Control (ACL) can be defined for each objects. This Access Control service protects critical resources in a system by limiting access only to authorized users and their applications. It prevents the unauthorized user of an object. For example, you can define access controls so that it only allows that particular application to connect to a queue manager only if the user ID associated with the application is authorized to do so.

► **Confidentiality**: many times you need to protect the message from unauthorized disclosure and you do not want to ignore the message content confidentiality when the message is travelling over an insecure network like Internet. In such cases, there is no help that we can get from access control definitions. What we need here is message encryption. For example, after sending the message MCA gets it from the transmission queue, the message is encrypted before it is sent over the network to the receiving MCA. At the other end of the channel, the message is decrypted before the receiving MCA puts it on its destination queue.

► **Data integrity** service describes more about unauthorized modification of the data. Such a modification of data is possible in two different cases, through hardware and transmission errors or because of deliberate attack.

Many hardware products and transmission protocols now have mechanism to detect and correct hardware and transmission errors. So, for our messaging security this may not be a threat or concern. But this is not the same with deliberate attacks.

Access control mechanism can contribute to data integrity to an extend as data cannot be modified if access is denied. So Data Integrity service can be used to detect whether the contents of the message have been modified while it was travelling over the network. This can also be helpful while messages are stored in a local queue, the access control mechanism provided by WebSphere MQ might be sufficient to prevent deliberate modification of the contents of the message. However, for a greater level of security, a data integrity service can be used to detect whether the contents of a message have been deliberately modified between the time the message was put on the queue and the time it was retrieved from the queue.

► **Non-repudiation** is more about providing with a proof of origin that the message was sent by one particular individual and providing with a proof of delivery that can provide the sender with undeniable evidence that the message was received by that particular individual.

For implementation neither IBM WebSphere MQ nor Tivoli Policy Director for MQSeries provides a non-repudiation as part of its base function. However can be achieved by writing your own exit programs within WebSphere MQ environment.

7.2.2 Messaging support for WebSphere Application Server

Messaging provider support for WebSphere Application Server V5.0 can be done mainly in three ways.

1. Using Embedded JMS Provider

2. External JMS provider WebSphere MQ V5.3

3. External Generic JMS providers

Embedded JMS provider does not have the same of security support compared to what we have in WebSphere MQV 5.3. The rest of this section will explore the security features for these two JMS Provider options with WebSphere Application Server.

> **Attention:** Message Driven Bean is a new functionality introduced in EJB 2.0 Specifications. A Message Driven Bean is a stateless component that is invoked by the container as a result of the arrival of a JMS message. Invoking the getCallerPrincipal and isCallerInRole methods is disallowed in the message-driven bean methods because the container does not have client security context. The Container will throw the java.lang.IllegalStateException if either of these methods is invoked.
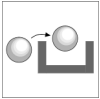
7.2.3 Security for WebSphere Embedded JMS Provider

This section discusses the security related features of the Embedded JMS Provider that comes with WebSphere Applicaiton Server V5.

Add/Remove queues to Embedded JMS Provider

The following steps describe, how to add and remove queues with the embedded WebSphere JMS provider.

1. In the navigation pane, select **Servers -> Application Servers**. Here you should be seeing the list of application servers.

2. In the content panel, click the name of the application server. In our case, it is **server1**, this displays the properties of the chosen application server.

3. In the content panel, under **Additional Properties**, select **Server Components -> JMSServer**, this displays the JMS properties.

4. If you want to add or remove queues, you can remove it from here. While adding you must follow the naming convention of WebSphere Embedded JMS Provider naming convention.

5. Save the configuration then restart the application server, to make the changes effective.

This creates the required queues in the Embedded JMS Provider for WebSphere.

Implement Security for WebSphere Embedded JMS Provider

The following steps describe, how to configure security for the embedded JMS provider.

1. Configure authentication settings owned by the internal JMS Provider. Authorization to access JMS resources owned by the internal JMS provider is controlled by settings in the XML file: <websphere_root>/config/cells/<your_cellname>/internal-jms-authorisations. xml. For queues and topics this file is the source of authorisation information. This file consists of which user has what permissions. For example,

internal-jms-authorisation.xml can grant read, write permissions on queues and Pub, Sub and Persist permissions on topics.

2. Edit the <queue-admin-userids> item in the XML file to create a list of user IDs who has administrative access over all queues. Administrative access is needed to create queues and perform other administrative operations on queues.

Example 7-6   Queue Administrators Configuration

```
<queue-admin-userids>
    <userid>quser01</userid>
    <userid>quser02</userid>
</queue-admin-userids>
```

In this example, quser01 and quser02 have administrative rights over all queues.

3. Edit <queue-default-permissions> section to define the default permissions for all queues. If this section is not defined then security constraints are valid only for those user IDs we define explicitly. This configuration will be overridden for a specific queue if <queue> section is created for that particular queue.

Example 7-7   Queue Default Permissions

```
<queue-dafault-permissions>
    <permission>read</permission>
</queue-default-permissions>
```

4. To define access permissions at the queue level, in the XML file create a <queue> node. and define the following elements.

   – <name> is the name of the queue.

   – <public> defines the default public access permissions. If you dont specify security constraints for a particular use, then these public attributes would apply.

   – <authorize> defines the access permissions for each user ID.

     • <userid> is the user ID you want to assign specific attributes.

     • <permission> is an access permission for the associated user ID.

Example 7-8   Queue Permissions

```
<queue>
<name>WQ_ITSOQueue</name>
    <public>
    </public>
    <authorize>
```

```
        <userid>quser01</userid>
        <permission>read</permission>
        <permission>write</permission
    </authorize>
    <authorize>
        <userid>quser02</userid>
        <permission>read</permission>
    </authorize>
</queue>
```

In the above example, user quser01 has both read and write access to the queue WQ_ITSOQueue where as user quser02 is having read-only permission to the messages in the queue.

5. To configure access permissions for the **topic** we need to update the <topic> section. In topic access permissions, inheritance of properties is applicable, which means that, if for a particular user access permissions are not explicitly specified then permissions are first inherited from the public permissions, then from parent of that topic and it continues till the root topic from which the root permissions are assumed.

   The structure of <topic> has the following elements:

   – <name> is the name of the topic.

   – <public> is the default access permissions for that topic.

   – <authorize> defines the access permissions for a specific user.

      • <userid> is the user that we want to assign access permissions.

      • <permission> is the type of access permission. The following permissions are applicable to a topic:

        +pub: Grant Publish permission.

        +sub: Grant Subscribe permission.

        +persist: Grant Persist permission.

        -pub: Deny Publish permission.

        -sub: Deny Subscribe permission.

        -persist: Deny Persist permission.

6. To define default permissions for a topic we need to configure topic node in the XML file as follows:

Example 7-9   Topic Default Permissions

```
<topic>
    <name></name>
    <public>
```

```
        <permission>+pub</permission>
    </public>
</topic>
```

Keeping the <name> element empty signifies that this configuration is the default topic configuration.

7. To define permissions for a specific topic we need to configure topic node in the XML file as follows:

Example 7-10   Topic Permissions

```
<topic>
    <name>ITSOTopic</name>
    <public>
        <permission>+pub</permission>
    </public>
    <authorize>
        <userid>tuser01</userid>
        <permission>+pub</permission>
        <permission>+sub></permission>
    </authorize>
</topic>
```

8. Save and close the internal-jms-authorizations.xml file.

Security for the WebSphere embedded JMS provider operates as part of WebSphere Application Server global security. That means to enable Embedded JMS Security we need to enable Global Security of the application server. When the security is enabled all the JMS connections to the JMS provider are all authenticated and access to JMS resources owned by JMS provider are controlled by access authentications. Also all JMS connections to JMS provider must provide user name and password for authentication.

We define this user name and password when we create a connection factory. In the administrative console,

1. Click **Resources -> WebSphere JMS Provider.**

2. In the Additional Properties click **WebSphere Queue Connection Factory**.

3. Click **New**. In this page of the Administrative Console set the following fields:

   – Component-managed Authentication Alias, select an authentication alias entry for this field. If no alias is available you need to create one.

   – Container-managed Authentication Alias, select an authentication alias entry for this field. If no alias is available you need to create one.

   If this authentication fails JMS Connections will not be created.

4. Save the configuration.
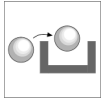
7.2.4 Security for WebSphere MQ (external provider)

This section will briefly describe how to manage security using an external messaging provider, in this case WebSphere MQ.

Use the Administrative Console of WebSphere Application Server for this configuration,

1. Navigate to **Resources -> WebSphere MQ JMS Provider,** click the **Additional Properties**.

2. Here you can configure JMS Queue Connection Factories and also Queue Destinations.

3. Select the scope for the configuration, in our case: **server**, then click **Apply**.

4. Click **MQSeries Queue Connection Factory**.

5. On the next page click **New**. Similarly to the Embedded JMS provider, specify the following fields beside the other non-security related fileds:

    – Component-managed Authentication Alias, select an authentication alias entry for this field. If no alias is available you need to create one.

    – Container-managed Authentication Alias, select an authentication alias entry for this field. If no alias is available you need to create one.

> **Important:** The user name and password specified here depends on the type of global security. Specify the OS user name and password when LocalOS is used as user registry; use an LDAP user name and password when using LDAP as user registry.

6. Click **Apply**.

7. Go back to **WebSphere MQ JMS Provider** and click **WebSphere MQ Queue Destinations**.

8. Click **New** which opens a new page, define the fields for the new queue destination.

9. Save the configuration.

Each time, a connection factory is created it first gets authenticated by the security information we have provided.
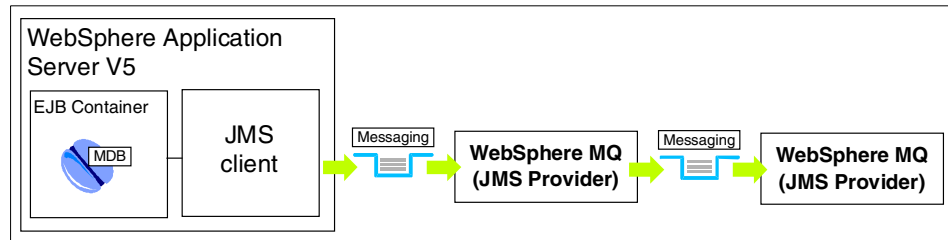
Figure 7-26   WebSphere Application Server and WebSphere MQ

As shown on the picture above, there are two areas where security is an issue for messaging.

1. Messaging security between two WebSphere MQ servers.

2. Messaging security between WebSphere Application Server, JMS Client and WebSphere MQ Server.

We are only going to discuss the security related to WebSphere Application Server, JMS Client and WebSphere MQ Server. Security between two messaging servers is out of the scope of this book. You can find all details about WebSphere messaging security in the IBM WebSphere MQ version 5.3 Security product documentation.

To administer WebSphere MQ the user should be a member of *mqm* group. The user ID *mqm* is created at the product installation time. On UNIX, all WebSphere MQ objects are owned by the user *mqm*. But on Windows platform members of Administrators group can also administer any Queue Manager.

Security Administrators add users to who need to administer WebSphere MQ to the mqm group. This includes the root user on UNIX systems.

Security checks are made for a typical application when connecting to the Queue Manager (**MQCONN and MQCONNX calls**), Opening an object (**MQOPEN and MQPUT1 calls**), putting and getting messages (**MQPUT and MQGET calls**) and closing the object (**MQCLOSE**).

Access Controls
Access controls can put restrictions on the user for authority to administer WebSphere MQ objects, authority to work with WebSphere MQ objects. When we are integrating WebSphere MQ with WebSphere Application Server then the application server should have all the previliges required to work with WebSphere MQ objects. On distributed platforms, the authorization service provides the access control when an application issues an MQ call to access a WebSphere MQ object that is a queue manager, queue etc. This includes check for alternate user authority and authority to set or pass context information.

The authority service component provided with WebSphere MQ is called the *Object Authority Manager (OAM)*. The OAM is automatically enabled for each queue manager. If you do not want any authority checks, you can disable the OAM.

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 appear in an ACL. This means that all members of a group have the same authority. On Windows both user IDs and group IDs can appear in an ACL. This means that authorities can be granted to individual users as well as groups. The control command `setmqaut` grants and revokes authorities and is used to maintain ACL. You can specify any number of authorizations in a single command. For example, the list of authorizations to permit a user or group to put messages on the queue and to browse them, but to revoke access to get messages is

```
+put +browse -get
```

The following example shows how to use the `setmqaut` command to grant and revoke permissions to use an object.

```
setmqaut -m QM1 -t queue -n ITSO.QUEUE -g ITSOGROUP +put +browse -get
```

In this example:

▶ `QM1` is the Queue Manager.

▶ `queue` is the object type.

▶ `ITSO.QUEUE` is the object name.

▶ `ITSOGROUP` is the identifier of the group whose authorizations are to change.

▶ `+put +browse -get` is the authorization list for the specified queue:
  - `+put` adds authorization to put (MQPUT) messages on the queue
  - `+browse` adds the authorization to browse messages on the queue (to issue MQGET with the browse option)
  - `-get` removes authorization to get (MQGET) messages from the queue.

SSL Support

Many times, it is required to secure data transmitting over an insecure network. WebSphere MQ supports SSL Version 3.0 on UNIX (installed with WebSphere MQ), Windows (Windows 200 has SSL support integral to the operating systems) and z/OS (SSL support is integral to the z/OS operating system).

Message channels and MQ channels can use the SSL protocol to provide link level security. A caller MCA is an SSL client and a responder MCA is an SSL server. You can specify the cryptographic algorithms that are used by the SSL protocol as part of the channel definition.

At each end of a message channel, and at the server end of an MQ channel, the MCA acts on behalf of the queue manager to which it is concerned. During the SSL handshake, the MCA sends the digital certificate of the queue manager to its partner MCA at the other end of the channel. The MCA at the client end of an MQI channel acts on behalf of the user of the WebSphere MQ client application. During the SSL handshake, the MCA sends the user's digital certificate to its partner MCA at the server end of the MQ channel.

Digital certificates are stored in a *key repository*. The queue manager attribute *SSLKeyRepository* specifies the location of the key repository that holds the queue manager's digital certificate. On a WebSphere MQ client system, the MQSSLKEYR environment variable specifies the location of the key repository that holds the user's digital certificate. Alternatively, a WebSphere MQ client application can specify its location in the KeyRepository field of the SSL configuration options structure, MQSCO, for an MQCONNX call.

To create your own certificates for the SSL communication, use IBM's ikeyman tool.

7.3 J2C Security

This section briefly describes the J2EE Connector Architecture in WebSphere Application Server V5 and the security considerations related to connectors.

When using connectors the application requests data from the connector and the connector gets the data from the resource and returns it to the application. But Enterprise Information Systems are generally very important applications and are protected from unauthorised access. So it requires authentication information to be passed while requesting a connection.
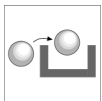
7.3.1 Securing adapters

Connectors in WebSphere let you connect to resources such as data or an application on a remote server. These resources are called as "Enterprise Information System" (EIS). Typically connector accesses non-relational data and is used by developers to developers to complement the other means of accessing Relational DataBase Management Systems (RDBMS) data. Basically, your application request reaches the connector, connector talks to the EIS, and return the results back to the requestor application.

J2EE Connector architecture establishes contracts amongst the application, the resource adapter and the application server, where the application will enentually be deployed. These contracts imply that all the participating components are J2EE Connector architecture compliant for the sake of working together. The application contract is nothing but the definition for the communication between connector and the application.

The system contract defines the connection management, transaction management and the security management.
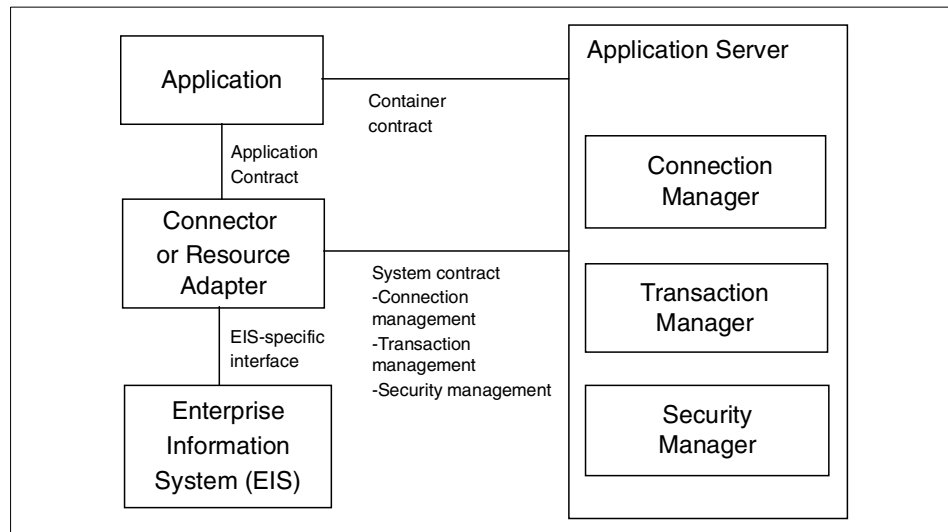

Figure 7-27   J2EE Connector Architecture

The security contract enables the application server to connect to an Enterprise Information System using security properties. The application server authenticates with the EIS system by using the security properties the user credentials.

There are two different methods the application server can authenticate to an Enterprise Information System.

► **Container-managed sign-on,** the security properties are configured when the resource adapter is deployed on the application server. Again there are several ways to configure security properties here. With *Configured Identity* all resource adapter connections use the same identity when connecting to the Enterprise Information System. With *Principal Mapping* the principal used when connecting to the Enterprise Information System based on a combination of the current principal in the application server and the mapping. With *Caller Impersonation* principal used in the Enterprise Information

System exactly matches the principal in the application server. With *Credentials Mapping* the type of credentials must be mapped from application server credentials to Enterprise Information System credentials.

Though it is easy to have container-managed sign-on, but there is only little flexibility as it is not possible to change the security properties in runtime.

► **Component-managed sign-on** allows you to pass security configuration properties each time a connection is acquired from the resource adapter.
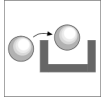
7.3.2 Java 2 Connector Security

Enterprise Information System stores very important information and the information must be protected from unauthorized users. Java 2 Connector architecture is designed to address the security of connection to Enterprise Information System. The application server and the Enterprise Information System collaborate to ensure the proper authentication of a resource principal which establishes a connection to an underlying entreprise information system. Connector architecture supports the following authentication mechanisms:

– BasicPassword: Basic username-password based authentication mechanism specific to enterprise information sysytem.

– Kerbv5: Kerberos version 5 based authentication model

WebSphere Application Server V5 Java 2 Connector supports basic password model currently. Kerberos authentication model will be supported in the near future.

The user ID and password for the target EIS is either supplied by applications or by the application server. WebSphere Application Server uses the JAAS pluggable authentication mechanism to perform principal mapping to convert WebSphere principal to resource principal. WebSphere Application Server provides a DefaultPrincipalMapping LoginModule, which basically converts any authenticated principal to the pre-configured EIS resource principal and password. Subsequently, you can plug in your own principal mapping LoginModule using the JAAS plug-in mechanism.

The user ID and password can either be configured using the Administrative Console or can be sent to the Enterprise Information System programmatically.

Using J2C Authentication Data Entries for Datasource

First you will have to create a new J2C entry for WebSphere. In order to create the appropriate entry for this section follow the configuration steps from Section 10.7.2, "J2C Authentication Data Entries" on page 256.

As a next step, the following list will explain how to configure a data source with J2C authentication in WebSphere Application Server V5 using the Administrative Console. This uses the J2C Authentication Data Entry created in the previous step.

1. In the Administrative Console navigate to **Resources -> JDBC Providers**.

2. Select the scope for the new provider, in our case **Server**, then click **Apply**.

3. Click **New** to create a new JDBC provider.

4. Choose **DB2 JDBC Provider** from the drop-down list provided and click **OK**.

5. In the DB2 JDBC Configuration page, make sure the classpath has the correct path for db2java.zip file.

6. Click **Apply**.

7. Open this panel once again, at the bottom of the page in **Additional Properties** section, click **Data Sources**.

8. Click **New**.

Figure 7-28   Data Sources configuration using J2C Authentication Data Entry

9. Create a J2C authentication entry for the new resource, following the steps from Section 10.7.2, "J2C Authentication Data Entries" on page 256.

10. Enter the following details from below:

Name is the name of the data source `itsobankds`.

JNDI Name is the JNDI Name for this resource `jdbc/itsobankds`.

Container Manager Persistence, if you want this data source to be used for the container managed persistence of EJBs then select this check box. This will create a corresponding CMP connection factory which corresponds to this datasource to be created for the relational resource adapter.

Component-managed Authentication Alias, in this drop-down list, you should see the J2C Authentication Data Entry that we created, which is `itsobankds_auth`. Select **itsobankds_auth** from the list.

Component-managed Authentication Alias, select `itsobankds_auth` from the list again.

11. Click **Apply**.

12. Save the configuration.

Connection Management for J2C adapters

In WebSphere Application Server V5 connection manager covers both JDBC and J2C connections.
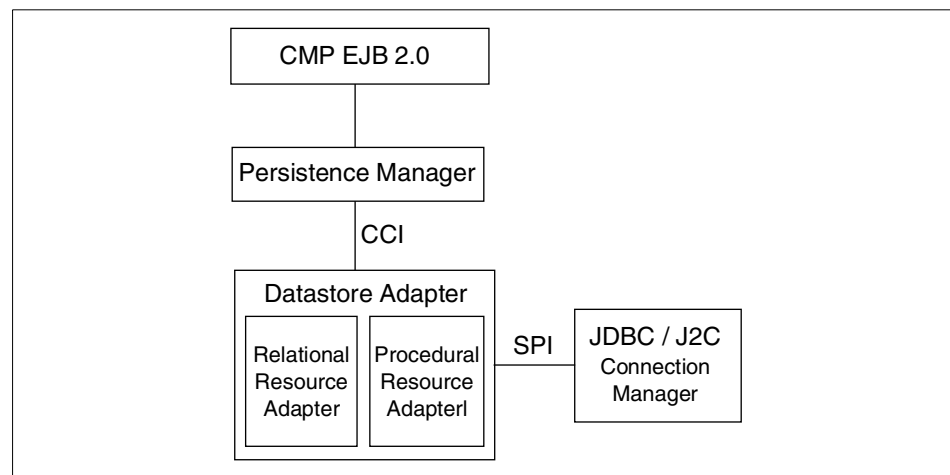
Figure 7-29   WebSphere Application Server V5 Connection Manager

The Persistence Manager uses the Common Client Interface (CCI) to connect to Resource Adapters, which provides access from J2EE clients, such as EJBs, JSPs, Servlets to an Enterprise Information System (EIS), so that the developer need not worry about the underlying semantics.

Resource Adapter is a system level software driver used by a Java application to connect to the Enterprise Information System. The resource adapter plugs into the application server to provide connectivity between application server and the Enterprise Information System.

JDBC usage of JCA connection manager

In WebSphere Application Server V5 JDBC Connection Manager architecture is slightly different from that of WebSphere Application Server V4.

EJB 2.0 Persistence Manager is based on J2C connections. It requires J2C style connection factory, interactions, cleanup and so on for JDBC connections. That means in WebSphere Application Server V5 a J2C Relational Resource Adapter is handling the JDBC connections.

WebSphere Application Server V5 supports both J2EE 1.2 and J2EE 1.3 applications.
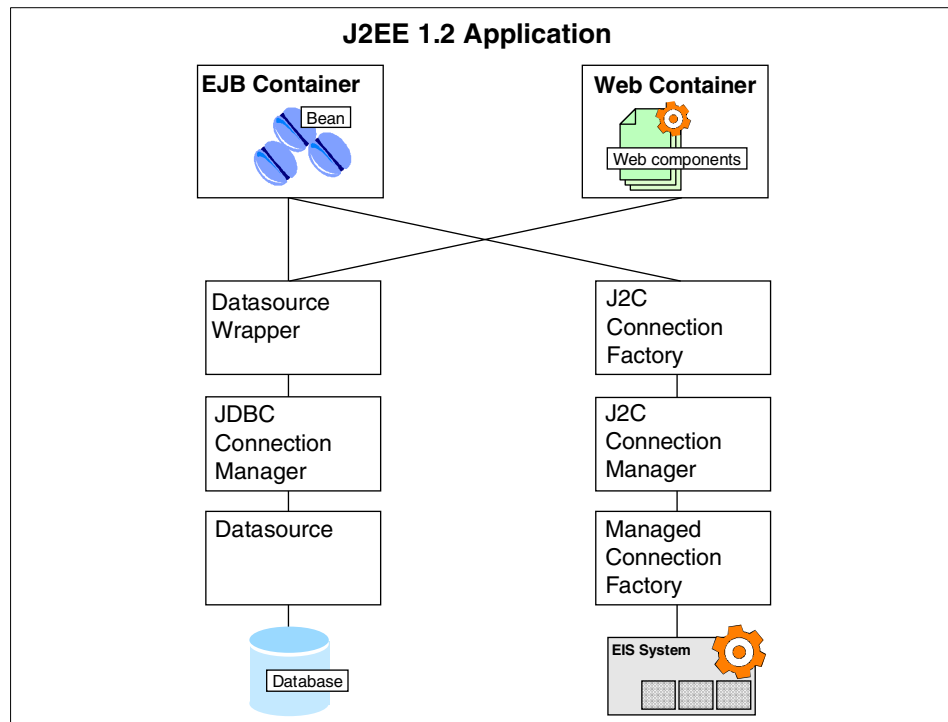
Figure 7-30   J2EE 1.2 Application

Between Figure 7-30 and Figure 7-31 the main difference is the introduction of generalised Connection Handle Manager and Connection Manager. Based on the type of request, Connection Manager will pass the request either to relational datasource or procedural connection factory for Enterprise Information Systems.
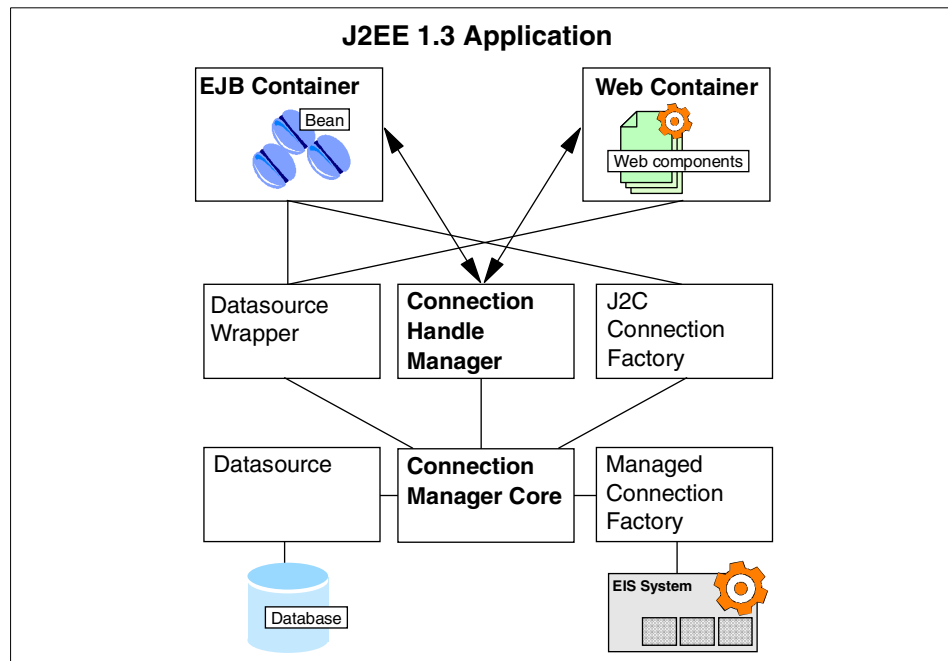
Figure 7-31   J2EE 1.3 Application

Accessing data using a JCA connectors

According to JCA specifications, each Enterprise Information System needs to implement a Resource Adapter and a Connection Factory.
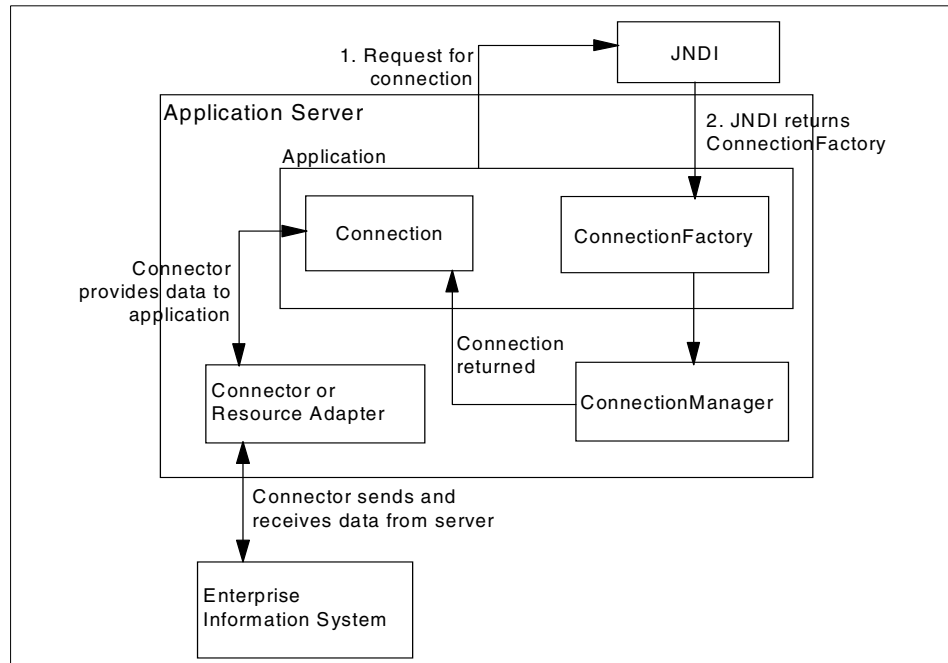
Figure 7-32   Connection to a resource

WebSphere Studio can generate the deployment descriptor and code for you.
For each connection follow the steps below:

1. Declare the connection factory resource reference in the application
    component's deployment descriptor.

Example 7-11   Connection factory resource reference in the deployment descriptor

```
<resource-ref>
   <description>description</description>
   <res-ref-name>eis/myConnection</res-ref-name>
   <res-type>javax.resource.cci.ConnectionFactory</res-type>
   <res-auth>Application</res-auth>
</resource-ref>
```

The *<res-auth>* element specifies whether the enterprise bean code signs on
programmatically to the resource manager, or the Container will sign on to the
resource manager on behalf of the bean. In the latter case, the Container
uses information that is supplied by the deployer. The value of this element
must be one of the two following:

<res-auth>Application</res-auth>

<res-auth>Container</res-auth>

2. Configure each resource adapter and associated connection factory through the Admin Concole.

3. In the application component, perform JNDI lookup to find the corresponding connection factory and get the connection.

Example 7-12   Accessing data using a JCA connector

```
javax.naming.InitialContext ctx = new javax.naming.InitialContext();
connectionFactory = (javax.resource.cci.ConnectionFactory)
ctx.lookup("java:comp/env/eis/myConnection");
// create a connection
connection = connectionFactory.getConnection();
// Create Interaction and an InteractionSpec
interaction = connection.createInteraction();
interactionSpec = new InteractionSpec();
interactionSpec.setFunctionName("GET");
// Create input record
inRec = new javax.resource.cci.Record();
// Execute an interaction
interaction.execute(interactionSpec, inRec, outRec);
// Process the output...
// close the interaction and connection
interaction.close();
connection.close();
```

4. Create an `InteractionSpec` object from the Connection object.

5. Create a Record object for the input / output data used by the functions.

6. Execute the functions through the Interaction object, process Record data and close the connection.

7.4 Where to find more information

For more information on the security aspects of J2EE, see the following documents:

▶ The Java 2 Platform Specification v1.3 at
   http://java.sun.com/j2ee/docs.html

▶ The specification for JCA (Java Connector Architecture) and JMS (Java Message Service) is also available at the previous URL.

▶ W3C Web site hosts most of the Web services related specifications, recommendateions and notices at http://www.w3c.org.

8

Programmatic security

Programmatic security comes handy when the application server provided security infrastructure cannot provide all the functionalities needed for the application.

Using the Java APIs for security can be the way to implement security for the whole application without using the application server security functions at all.

Programmatic security also gives the option to implement dynamic security rules for your applications.

8.1 Programmatic security

J2EE security can be applied declaratively or programmatically. This chapter will focus on the latter. Programmatic security can be used by security aware applications when declarative security alone is not sufficient to express the security model of the application.

As an example, the ITSOBank application supplied with this book is configured such that only managers and employees (clerks and accountants) can transfer funds but anyone could check their balance. This is possible because the method permissions for the getCustomerBalance method on the Consultation EJB allows the necessary role (in this case, Consultant) access. The request simply passes the account key as a parameter.

8.2 J2EE API

WebSphere provides a security infrastructure for application security which is transparent to the application developer. That is, the developer does not need to code for security, since it will all be handled at deployment and runtime.

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.

8.2.1 EJB security methods

The EJB 2.0 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, you need to call getName() on the java.security.Principal object returned.

```
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.

```
EJBContext ejbContext;
...
if (ejbContext.isCallerInRole(""))
    // Perform some fuction
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 will return `null`.

Sample usage of security methods

The following example is a modified code snippet from the ITSOBank sample application. You can find the similar code in the TransferBean.java in the transferBranch2Customer() method. For more details check the comments in the source below, or in the original sample application.

Example 8-1   Sample code using the EJB security methods

```
// getting the environment variables for restricted role
// and for maximum transferable amount
restrictedRole=(String)environment.lookup("RestrictedRole");
maxEJBTransferAmount=(Integer)environment.lookup("MaxEJBTransferAmount");
// checking if the user is restricted to a certain amount of transfer
if(mySessionCtx.isCallerInRole(restrictedRole) &&
transferAmount>maxEJBTransferAmount.intValue()) {
   // the user cannot transfer the requested amount
   return false;
}
// get the caller principal, then the user name
java.security.Principal callerPrincipal=mySessionCtx.getCallerPrincipal();
String callerName =callerPrincipal.getName();
// print out the user information about the EJB method invocation
System.out.println("... method was invoked on the Transfer EJB by:
"+callerName);
```

With the security methods the EJB will not let the user in a restricted role to submit a transfer greater than the maximum transferable amount.

8.2.2 Servlet security methods

The Servlet 2.3 specification defines three methods that allow programmatic access to the caller's security information of HttpServletRequest interface.

> **Important:** The methods *getRemoteUser()* and *getUserPrincipal()* 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 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 which are not possible, or 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, you need to call getName() on the java.security.Principal object returned.

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

Sample usage of security methods

The following example is a modified code snippet from the ITSOBank sample application. You can find the similar code in the TransferServlet.java in the doPost() method. For more details check the comments in the source below or in the sample application.

Example 8-2   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 will not let the user in a restricted role to submit a transfer greater than the maximum transferable amount.

8.3 CustomRegistry SPI

WebSphere supports the use of user registries in order to lookup user and group details for authentication purposes. Three registries are provided by default, although only two are likely to be commonly used. These are the local OS registry, an LDAP server and a filesystem-based registry called FileRegistrySample. The FileRegistrySample registry is not to be used in production environments due to its lack of scalability, but is included as an example of how a custom registry might operate. In fact, it is possible to develop integration with any type of custom registry that supports the notion of users and groups by implementing WebSphere's UserRegistry interface. The UserRegistry interface is provided so that the application server may make use of a user registry that would otherwise be inaccessible. This interface is defined in the com.ibm.websphere.security package.

The provision of this interface ensures that a variety of user registries may be used such as relational databases, files stored on directly on the filesystem or integration products such as WebSphere MQ. A combination of multiple registries may be used such as LDAP and RACF. A demonstration custom registry that uses flat files as the data store is supplied with the application server.

The UserRegistry interface defines a general set of methods to allow the application server to obtain user and group information from the regsitry. The registry can operate as a process running remotely to the application server and so it is necessary for each registry to implement the java.rmi.Remote interface.

One point worth noting regard to the difference between the initialisation of a WebSphere Application Server V5 custom registry and a WebSphere Application Server V4 custom registry. With Version 4, it was possible to use other WAS components to initialise the custom registry. For example, a datasource might have been used to connect to a database-based custom registry or one may have made use of a deployed EJB. However, in Version 5, both of these examples are not possible because, unlike in version 4, the security mechanism is initialised before other components such as containers and therefore these facilities are not available when the security component is started.
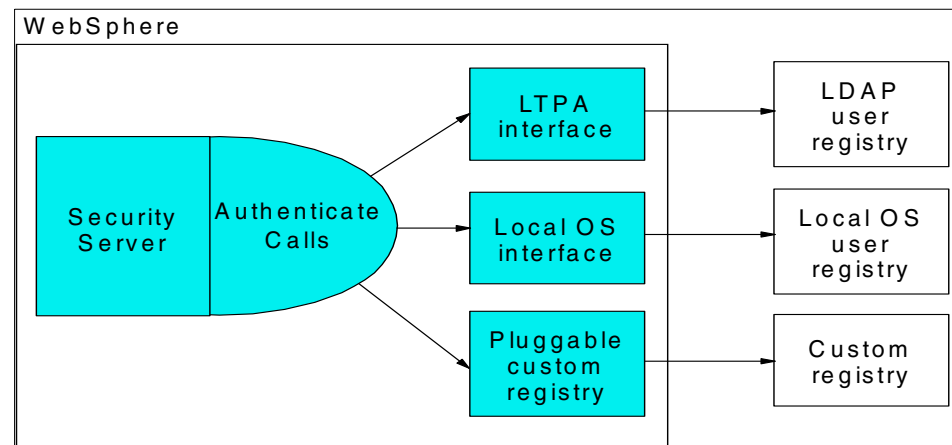
Figure 8-1   Authentication registry mechanisms

Developing a custom registry

A sample custom registry implementation is provided with the application server. The custom registry class is called FileRegistrySample. The source code is also provided for reference purposes. Refer to the InfoCenter for details on configuring the application server to use this.

Some points should be taken into consideration when developing a custom registry.

► The `com.ibm.websphere.security.UserRegistry` interface must be implemented. By implementing this interface, the super-interface, `java.rmi.Remote`, will also be implemented

► The implementation should be compiled by adding the wssec.jar and idl.jar files to the classpath. These files can be found in the <WAS_HOME>/lib directory. The following command should be used to compile the registry

    <WAS_HOME>/java/bin/javac -classpath
    <WAS_HOME>/lib/wssec.jar:<WAS_HOME>/lib/idl.jar <source_file>

This will generate some classes that should be referenced in the application server's classpath. Alternatively, move the classes to the <WAS_HOME>/classes directory.

The application server must be configured to make use of the custom registry.

1. From the navigation panel in the Administrative Console, select Custom User Registry under User Registries in the Security Center.
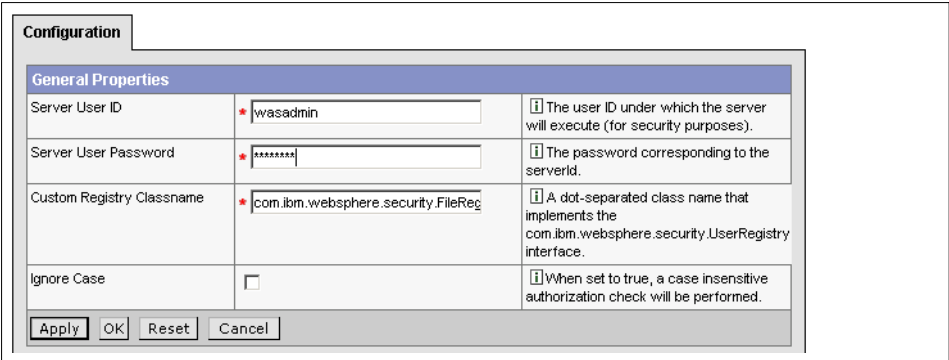
Figure 8-2   User registry configuration

2. Enter the user name and password of an identity under which the custom registry will operate. Enter the class name for the custom registry. For example, the supplied sample registry is `com.ibm.websphere.security.FileRegistrySample`. Click **Apply.**

3. Click **Custom Properties** and add the properties necessary to initialise the registry. These properties will be passed to the initialize method of the custom registry. For the supplied FileRegistrySample code, enter the following properties.

Table 8-1   FileRegistrySample initialisation properties

Name	Value
usersFile	users.props
groupsFile	groups.props
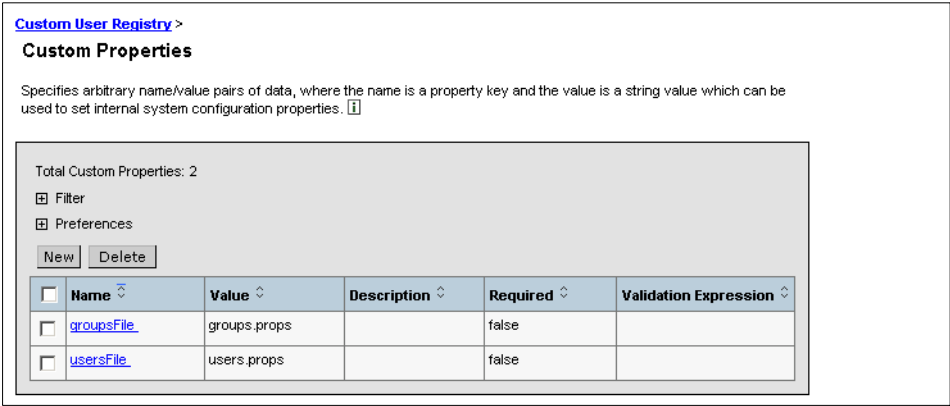
Figure 8-3   Custom Registry properties

> Refer to the Infocenter, for details regarding the format of these files.

4. Be sure to save the changes to the master configuration before proceeding.

5. In the case of WebSphere Application Server Network Deployment, the custom registry class, classpath settings and properties will need to be applied to all of the application servers in the cell. The FileRegistrySample registry is already installed on each application server and so does not need distributing.

6. To activate the custom registry, Global Security must be enabled. Go to the Global Security panel, click the **Enabled** check box and ensure the Active User Registry is set to **Custom**. Click **OK**.

7. If the validation passes then the changes should be saved to the master configuration and the server restarted for the security changes to take effect.

The UserRegistry interface

The list below includes all the methods defined in the UserRegistry interface. Each method must be implemented by the custom registry.

Table 8-2   WebSphere's 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.

Method signature	Use
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 X509 format) to a valid user in the Registry. This is used to map the name in the certificate supplied by a browser to a valid userSecurityName in the registry.
String getRealm() throws CustomRegistryException, RemoteException	The realm is a registry-specific string indicating the *realm* or *domain* for which this registry applies. For example, for OS400 or AIX this would be the host name of the system whose user registry this object represents. If null is returned by this method realm defaults to the value of "customRealm".
Result getUsers(String pattern, int limit) throws CustomRegistryException, RemoteException	Gets a list of users that match a *pattern* in the registy. The maximum number of users returned is defined by the *limit* argument.
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.

Chapter 8. Programmatic security   **187**

Method signature	Use
boolean isValidUser(String userSecurityName) throws CustomRegistryException, RemoteException	Determines if the *userSecurityName* exists in the registry.
Result getGroups(String pattern, int limit) throws CustomRegistryException, RemoteException	Gets a list of groups that match a *pattern* in the registy. The maximum number of groups returned is defined by the *limit* argument.
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 Unique id for a group.
List getUniqueGroupIds(String uniqueUserId) throws EntryNotFoundException, CustomRegistryException, RemoteException	Returns the Unique ids for all the groups that contain the UniqueId of a user. Called during creation of a user's credential.
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 *groupSecurityName* 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 *limit* argument.

Method signature	Use
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	Throw the NotImplementedException for this method.

Sample custom registry

The following documents the implementation of a DB2 custom registry. The DB2 registry uses JDBC to communicate with the database. Although this registry was tested with DB2, it should be possible to modify it to work with other relational databases. The source code (DB2Registry.java) is shown below and is also included in the ITSOBank application, along with the database structure.

Open the DB2Registry.java source in WebSphere Studio and check the comments in the source code. You will find all the required methods for the UserRegistry interface implemented. Look for the SQL queries in the code and see what each method does with the database,

The DB2Registry.class file must be copied to a directory accessible by the application server, that is a directory that is in the application server's classpath. For example, <WAS_HOME>/lib. Alternatively, update the application server's classpath to refer to the directory that contains the class file.

A simple custom registry test utility that runs from the command line is included which can be used to test if 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. The tool will ask for some user and group information and use this information to query the custom registry. It will also ask for a X.509 certificate file, although the response can be empty (just press **Enter**). In this case, the certificate check will not be made. For details on creating a digital certificate, refer to Section 10.9.1, "Generating a self-signed certificate" on page 263.

8.4 Custom Trust Association Interceptor

The application server can be configured to use a third-party product to provide authentication services, whilst continuing to perform authorisation. These products are often referred to as reverse proxy servers. To delegate the role of authentication to a reverse proxy, two conditions must be met.

► The reverse proxy must provide a Trust Association Interceptor, which WebSphere will use to receive requests from the reverse proxy server.

► A trust association between WebSphere and the reverse proxy must be established.

In order to provide an interceptor, the `com.ibm.websphere.security.TrustAssociationInterceptor` interface, which defines three methods, must be implemented.

► `public boolean isTargetInterceptor(HttpServletRequest) throws com.ibm.websphere.security.WebTrustAssociationException`

Determines whether the request originated with the proxy server associated with the interceptor. The implementation code must examine the incoming request object and determine if the proxy server forwarding the request is a valid proxy server for this interceptor.

► `public void validateEstablishedTrust(HttpServletRequest) throws com.ibm.websphere.security.WebTrustAssociationFailedException`

Determines if the proxy server from which the request originated is trusted or not. This method is called after the isTargetInterceptor method. The implementation code must authenticate the proxy server. The authentication mechanism is proxy-server-specific.

► `public String getAuthenticatedUsername(HttpServletRequest) throws com.ibm.websphere.security.WebTrustAssociationUserException`

The application server has accepted the proxy server's authentication of the request and must now authorise the request. This method extracts the request's user name from the HTTP header to allow for authorisation.

Configuring the Interceptor

In order to make an inteceptor configurable, it is necessary for it to extend `com.ibm.websphere.security.WebSphereBaseTrustAssociationInterceptor`. Three methods must be implemented.

► `public int init(java.util.Properties)`

Accepts a Properties object which contains the necessary interceptor configuration information.

► `public int init(String)`

Accepts a filename for a file that contains the necessary interceptor configuration information. The format of the properties file is specified in the Infocenter.

► `public void cleanup()`

Prepares the interceptor for termination.

Configuring WebSphere to use the Interceptor

Once the interceptor has been installed and the trust association is configured, the application server may receive requests from the reverse proxy server. The authentication method used must be LTPA and authentication cannot be delegated to a reverse proxy server if the user registry is LocalOS. For information regarding the configuration of WebSphere for LTPA, refer to Section 10.6, "LTPA" on page 249 and Trust Association Interceptors, refer to Section 12.5.1, "Single Sign-On with WebSEAL" on page 382.

Note: The InfoCenter provides information regarding the setup of Tivoli WebSEAL as a reverse proxy and the application server includes an implementation of the TAI for this product. Also, refer to Section 12.6.1, "Tivoli WebSEAL" on page 409.

Custom Trust Association Interceptor

This section will provide information on how to develop your own Trust Association Interceptor, how to configure a new interceptor for WebSphere and finally how to test it.

Important: The custom trust association interceptor here is only provided to show how to develop a custom interceptor. The interceptor provided here is not secure enough to use in any real environment.

Developing the custom Trust Association Interceptor

The Trust Association Interceptor (TAI) for this book was developed using the WebSphere Studio.

Required libraries from WebSphere for development: j2ee.jar, security.jar, securityimpl.jar, wssec.jar.

This sample is a sub-class of the WebSphereBaseTrustAssociationInterceptor, for more information about the code open the CustomTAI,java source in WebSphere Studio and check the comments. You will find all the required methods implemented for the TrustAssociationInterceptor interface.

For more information on Custom Trust Association Interceptors refer to the article on the IBM developerWorks Web site, Third-party security servers and WebSphere at http://www-106.ibm.com/developerworks/library/it-expertprog_tpss/index.html.

Configuring the custom Trust Association Interceptor

For testing purposes this example will use Tivoli Access Manager WebSeal as a security reverse proxy.
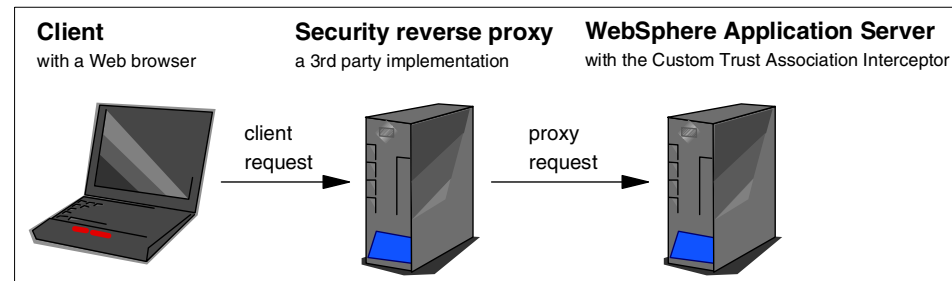

Figure 8-4   Environment for testing the custom trust association interceptor

The following configuration steps assume that you have already enabled global security, set the authentication mechanism to LTPA, and enabled Single Sign-On.

For testing purposes it is recommended to turn on the tracing facility for WebSphere, in order to do that follow the steps from Section , "Security trace" on page 235.

1. Move the .jar file with the custom association interceptor to the <WebSphere_root>/classes directory, next time you restart the server, it will pick up the .jar file and insert it into the classpath.

2. Register the new interceptor with the Administrative Console, navigate to the **Security -> Authentication Mechanisms -> LTPA** item.

3. Select **TrustAssociation**, then click **New.**

4. Provide the name for the class, in this case: **com.ibm.itsobank.tai.CustomTAI.**

5. Click **OK**.

6. Select the new interceptor, then click the **Custom Properties** link.

7. Create the following custom properties from the table below.

Table 8-3   Custom properties for the interceptor

Property name	value
proxyserver	wsl01
proxyport	443
proxyuser	wsl01user
proxypassword	password

Save the configuration for WebSphere to make the changes effective.

8. You will have to modify the application to use basic authentication for the Web application. You can either export the ITSOBank application, modify with the Application Assembly Tool, then redeploy it; or stop the server open the deployed application with the Application Assembly Tool (AAT) by selecting the ITSOBank.ear directory in AAT, do the modification, then restart the server.

> **Note:** This custom trust association interceptor only works with basic authentication.

9. Restart the application server.

10. Create a junction for the WebSeal proxy using the following command in the *pdamin* administration client:

```
server task webseald-wsl01 create -t ssl -h appsrv01 -p 9443 -B -U
"wsl01user" -W "password" -c all /customtai
```

Where `wsl01` is the name of the proxy server, `appsrv01` is the name of the WebSphere application server, `9080` is the port number for the embedded WebSphere HTTP server, `wsl01user` is a user registered with the `password` in the user registry that will be used to authenticate the proxy server to the application server.

> **Note:** You need to create the `wsl01user` in the user registry and set the password to `password`.

Testing the custom Trust Association Interceptor

Open a browser on the client machine, then access the ITSOBank application via the proxy at the following address: https://wsl01/customtai/itsobank

Provide the user name and password for a valid ITSOBank user, for example `manager01` and `password`.

Access the Customer transfer link, then submit a transfer. If you can access the page with the form with the transfer details that means, the custom trust association interceptor is working.

To make sure that the interceptor was working, open the trace.log file in the <WebSphere_root>/logs/server1 directory, then search for the **CustomTAI** string. After the request for the customertransfer.html page, you should find something similar to the following trace:

Example 8-3   Invocation of the custom trust association interceptor

```
[9/25/02 18:03:14:312 EDT] 41ab0df5  d UOW=
source=com.ibm.ws.security.web.WebAuthenticator org=IBM prod=WebSphere
component=Application Server
        handleTrustAssociation
[9/25/02 18:03:14:312 EDT] 41ab0df5  d UOW=
source=com.ibm.ws.security.web.WebAuthenticator org=IBM prod=WebSphere
component=Application Server
        TrustAssociation is enabled.
[9/25/02 18:03:14:312 EDT] 41ab0df5  > UOW=
source=com.ibm.ws.security.web.TrustAssociationManager org=IBM prod=WebSphere
component=Application Server
        getInterceptor
[9/25/02 18:03:14:312 EDT] 41ab0df5  d UOW=
source=com.ibm.ws.security.web.TrustAssociationManager org=IBM prod=WebSphere
component=Application Server
        Check if target interceptor ...
[9/25/02 18:03:14:312 EDT] 41ab0df5  O UOW=  source=SystemOut org=IBM
prod=WebSphere component=Application Server
        CustomTAI : isTargetInterceptor invocation
[9/25/02 18:03:14:312 EDT] 41ab0df5  O UOW=  source=SystemOut org=IBM
prod=WebSphere component=Application Server
        CustomTAI : via header:HTTP/1.1 wsl01:443
[9/25/02 18:03:14:312 EDT] 41ab0df5  O UOW=  source=SystemOut org=IBM
prod=WebSphere component=Application Server
        CustomTAI : request via host:wsl01 ,request via port:443
[9/25/02 18:03:14:312 EDT] 41ab0df5  O UOW=  source=SystemOut org=IBM
prod=WebSphere component=Application Server
        CustomTAI : request is coming through our proxy, we can accept it
[9/25/02 18:03:14:312 EDT] 41ab0df5  d UOW=
source=com.ibm.ws.security.web.WebAuthenticator org=IBM prod=WebSphere
component=Application Server
        A TrustAssociation interceptor is available for this request.
[9/25/02 18:03:14:312 EDT] 41ab0df5  O UOW=  source=SystemOut org=IBM
prod=WebSphere component=Application Server
        CustomTAI : validateEstablishedTrust invocation
[9/25/02 18:03:14:312 EDT] 41ab0df5  O UOW=  source=SystemOut org=IBM
prod=WebSphere component=Application Server
        WebCollaborator.pnAuthorization:Authorization
```

```
[9/25/02 18:03:14:312 EDT] 41ab0df5  O UOW=  source=SystemOut org=IBM
prod=WebSphere component=Application Server
        CustomTAI : request user name:wsl01user ,request password:password001
[9/25/02 18:03:14:312 EDT] 41ab0df5  O UOW=  source=SystemOut org=IBM
prod=WebSphere component=Application Server
        CustomTAI : authentication for the proxy was successful
[9/25/02 18:03:14:312 EDT] 41ab0df5  d UOW=
source=com.ibm.ws.security.web.WebAuthenticator org=IBM prod=WebSphere
component=Application Server
        TrustAssociation has been validated successfully.
[9/25/02 18:03:14:312 EDT] 41ab0df5  O UOW=  source=SystemOut org=IBM
prod=WebSphere component=Application Server
        CustomTAI : getAuthenticatedUsername invocation
[9/25/02 18:03:14:312 EDT] 41ab0df5  O UOW=  source=SystemOut org=IBM
prod=WebSphere component=Application Server
        CustomTAI : user name picked-up:manager
[9/25/02 18:03:14:312 EDT] 41ab0df5  d UOW=
source=com.ibm.ws.security.web.WebAuthenticator org=IBM prod=WebSphere
component=Application Server
        Username retrieved is [manager]
```

As you follow the trace from the beginning you will find how the trust association interceptor handles the request and retieves the user name from the original request.

8.5 Java 2 Security

The earlier Java implementations, prior to Java V1.2, only had the sandbox model, which provided a very restricted environment. In Java V1.2 a new security model has been introduced.

For more information refer to the official Java Sun site at http://java.sun.com/security/index.html, or the Java 2 Platform Security Architecture V1.0 paper from Sun.

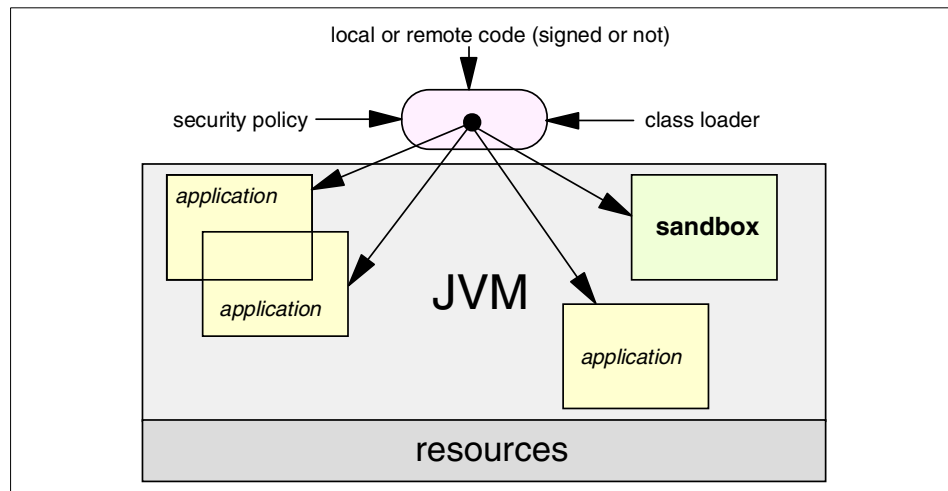The following diagram depicts the new security model for Java V1.2.

Figure 8-5   Java 2 Platform Security Model

The new model supposed to provide the following security features for the Java Virtual Machine:

- ► Fine-grained access control. It was available in earlier version using programmatic access control security.

- ► Easy configuration of security policy. It was available also as the previous features, and again using programmatic security.

- ► Easy extension for the access control structure. The new architecture allows typed security permissions and provides automatic handling for them.

- ► Extension of security checks to all Java programs (both applications and applets). Every Java code is under security control, which means no local code is trusted by default anymore.

The fundamental concept and an important building block in system security is the protection domain.

> **Definition:** A *domain* can be scoped by the set of objects that are currently directly accessible by a principal, where a principal is an entity in the computer system to which permissions are granted.
>
> Classes that have the same permissions but are from different code sources belong to different domains.
>
> (From the Java 2 Platform Security Architecture V1.0 paper by Sun Microsystems)

> **Definition:** A *principal* is an entity in the computer system to which permissions (and as a result, accountability) are granted.
>
> (From the Java 2 Platform Security Architecture V1.0 paper by Sun Microsystems)

There are two distict categories of protection domains:

► system domain
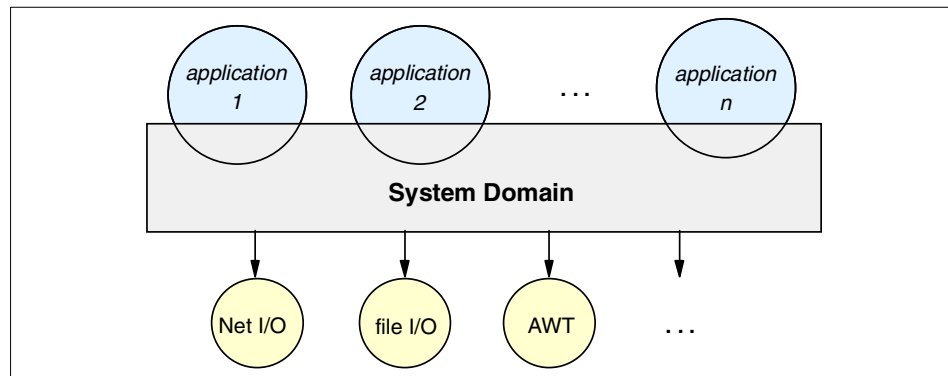
► application domain

Figure 8-6   Protection domains

Protection domains are determined by the policy currently in effect. The Java application environment maintains the mapping between code, their protection domains and their permissions.
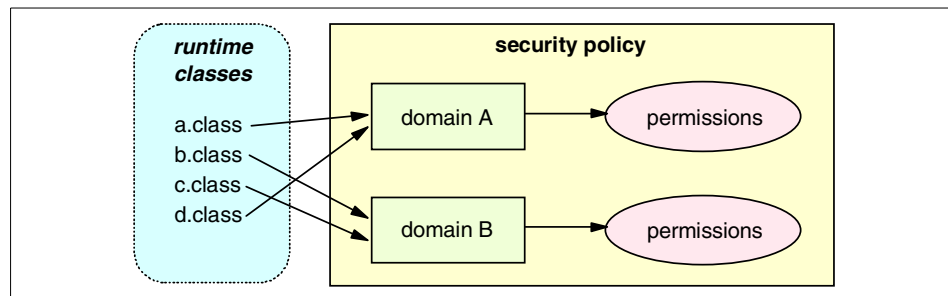

Figure 8-7   Class - Domain - Permission mapping

As a rule of thumb, a less "powerful" domain cannot gain additional permissions as a result of calling or being called by a more "powerful" domain.

To call a piece of trusted code to temporarily enable access to more resources than are available directly to the application the *doPrivileged* method can be used. It comes handy when for example an application wants to ....

The domains, either system or application, can also implement additional protection of their internal resources within the domain boundary.

Security management

The security manager defines the outer boundaries of the Java sandbox. The fact that the security manager is customizable, it allows the security manager to establilish custom security policies for an application. The concrete *SecurityManager* provided with the Java version 1.2 allows you to define your custom policy not in Java code, but in an ASCII file called the *policy file*.

The security manager is not automatically loaded when an application is running, in order to activate the manager, the user has to specify the *-Djava.security.manager* command-line argument for the Java runtime.

A custom security manager class can be also specified in the command-line: *-Djava.security.manager=com.mycompany.MySecurityManager*; if nothing is specified then the default security manager will be initialized for the application.

Access control

The *java.security.ProtectionDomain* class represents a unit of protection within a Java application environment and is typically associated with a concept of *principal*.

The java.security.AccessController class is used for the following purposes:

► to decide whether an access to a critical resource is allowed or denied, based on the security policy currently in effect,

► to mark code as being *privileged*,

► to obtain a *snapshot* of the current calling context to support access-control decisions from a different context.

Any code that controls access to system resources sould invoke AccessControler methods if it wishes to use the specific security model and access control algorithm utilized by these methods.

Security permissions

The permission classes represent access to system resources. The java.security.Permission class is an abstract class and is subclassed to represent specific accesses.

The list of permissions in Java V1.2:

▶ java.security.Permission

This abstract class is the anchestor of all permissions.

▶ java.security.PermissionCollection

It holds a collection of the same type of permissions (homogeneous).

▶ java.security.Permissions

It holds a collection of any type of permissions (heterogeneous).

▶ java.security.UnresolvedPermission

When the policy is initialized and the code that implements a particular permission has not been loaded or defined in the Java application environment; in this case the UnresolvedPermission holds the "unresolved" permissions.

▶ java.security.UnresolvedPermissionCollection

It holds a collection of UnresolvedPermission.

▶ java.io.FilePermission

Holds permission definitions for file resources, actions on a file can be: read, write, delete and execute.

▶ java.security.SocketPermission

This permission represents access to network sockets; actions on a socket can be: accept, connect, listen, resolve.

▶ java.security.BasicPermission

It extends the Permission class and can be used as the base class for other permissions.

▶ java.util.PropertyPermission

This class targets the Java properties as set in various property files; actions can be: read, write.

▶ java.lang.RuntimePermission

The target for this permission can be represented by any string and there is no action associated with the targets. For details on the targets, refer to Sun's Java 2 Platform Security Architecture document.

▶ java.awt.AWTPermission

Similar to the previous permission, but it is related to targets in AWT. For details on the targets, refer to Sun's Java 2 Platform Security Architecture document.

▶ java.net.NetPermission

It controls the Net related targets, no actions associated. For details on the targets, refer to Sun's Java 2 Platform Security Architecture document.

► java.lang.reflect.ReflectPermission

This is a Permission class for reflective operations; it has no actions; it works like the RuntimePermission. For details on the targets, refer to Sun's Java 2 Platform Security Architecture document.

► java.io.SerlializablePermission

It controls the serialization related targets, no actions associated. For details on the targets, refer to Sun's Java 2 Platform Security Architecture document.

► java.security.SecurityPermission

It controls access to security related objects; no actions associated. For details on the targets, refer to Sun's Java 2 Platform Security Architecture document.

► java.security.AllPermission

This permission implies all permissions.

> **Important:** no one except Sun Microsystems should extend the permissions that are built into the Java 2 SDK.
>
> (From the Java 2 Platform Security Architecture V1.0 paper by Sun Microsystems)

Policy files

The policy can be specified within one or more policy configuration files; where the files indicate what permissions are allowed for codes from specified code sources.

> **Definition:** a policy configuration file essentially contains a list of entries. It may contain a *keystore* entry, and contains zero or more *grant* entries.

The keystore can be defined according to the following grammar:

```
keystore "keystore_URL", "keystore_type";
```

A grant entry can be defined according to the following grammar:

```
grant [SignedBy "signer_names"] [, CodeBase "URL"] {
   permission permission_class_name [ "target_name" ] [, "action"]
[,SignedBy "signer names"];
   ...
};
```

Each grant entry consists of a CodeSource and its permissions, where a CodeSource conists of a URL and a set of certificates and the grant entry includes a URL and a list of signer names.

Property expansion is possible in the policy files and in the security properties file.

Example 8-4   Sample policy file

```
keystore "c:\keystores\mykey.jks", "jks"

grant codeBase "http://java.sun.com/*", signedBy "WebDeveloper" {
    permission java.io.FilePermission "/files/*", "read";
    permission java.io.FilePermission "${user.home}", "read,write";
}
```

When the JVM is loading a new class, the following algorithm is used to cehck the policy settings for that particulas class:

1. Match the public keys, if code is signed

2. If a key is not recognized in the policy, ignore the key. If every key is ignored, treat the code as unsigned.

3. If the keys are matched, or no signer was specified, try to match all URLs in the policy for the keys.

4. If either key or URL is not matched, use built-in default permission, which is the original sandbox permission.

Policy files in runtime
The following list will show, how the policy files can be specified for a Java runtime; and where those policy files are located.

▶   System policy file is located at:

   ```
   {java.home}/lib/security/java.policy
   ```

▶   User policy file is located at:

   ```
   {user.home}/.java.policy
   ```

▶   Policy file locations are also specified in the security properties file, located at:

   ```
   {java.home}/lib/security/java.security
   ```

▶   It is also possible to specify additional or different policy file when invoking execution of an application, using the appropriate command line arguments, for example:

   ```
   java -Djava.security.manager -Djava.security.policy=MyPolicyURL
   MyApplication
   ```

When the policy file is specified by using double equals, then the specified policy file will be used exclusively; for example:

```
-Djava.security.policy==MyOnlyPolicyURL
```

Security Exceptions

The following exceptions ship with the Java V1.2 SDK:

▶ java.security.SecurityException

This exception and its subclasses should be runtime exceptions (unchecked, not declared) that are likely to cause the execution of a program to stop. Such an exception is thrown when security violation is detected. For example: trying to access an unauthorized resource.

▶ java.security.GeneralSecurityException

This is a subclass of java.lang.Exception (must be declared or caught) that is thrown in other cases. Such an exception is thrown when security related but not vital problem is detected. For example: passing an invalid key.

Secure class loading

The dynamic class loading is one of the strength of the Java platform, because it provides the ability to install components at runtime. It is also critical in providing security because the class loader is responsible for locating and fetching the class file, consulting the security policy, and defining the class object with the appropriate permissions.

The *java.security.SecureClassLoader* is a subclass and an implementation of the abstract *java.lang.ClassLoader* class. Other classloaders subclass the SecureClassLoader to provide different class loading facilities for various applictions.

Debugging security

Use the *-Djava.security.debug=access,failure* argument in the virtual machine. This flag will dump the name of permission checks that are failing.

For example: start with minimal security permissions, then run a test and check what permissions are failing. Add the necessary permissions to the policy file then run your test again for re-checking. Repeat these steps until you have all the necessary permissions set. Note that thiss will only help you to identify the persmissions you have to set, it will not help to find the right settings for the permissions.

Security Tools for Java

The Java 2 SDK provides three tools that assist in the deployment of the new security features.

For more information refer to the documents under the Java 2 SDK directory in /docs/tooldocs/.

Key and certificate management tool

keytool is a key and certificate management utility, similar to IBM's ikeyman utility. Tha major difference is that the keytool is only a command line utility without graphical interface; but keytool allows to create certificates for any Distinguished Name (dn) that you require; unlike ikeyman which has a predefined *dn* schema. For online help launch the keytool with the -help option:

```
keytool -help
```

Policy file editing tool

The *policytool* with a nice GUI assists the user in specifying, generating, editing, exporting or importing a security policy. The application can be launched from the command line with the **policytool** command.

The policytool utility depends on the keystore that is managed by keytool.

JAR signing and Verification tool

The *jarsigner* tool can be used to digitally sign Java archives (JAR files) and to verify such signetures. The jarsigner tool can be used from the command-line by isuing the **jarsigner** command.

The jarsigner tool depends on the keystore that is managed by keytool.

8.5.1 Java 2 security in WebSphere

The WebSphere Application Server V5 also supports the Java 2 security in order to harden the Java Virtual Machine runtime environment.

By default, WebSphere Application Server installs a Java 2 SecurityManager and Java 2 Security is enforced via a default set of policies. The default policies are those recommended in the J2EE Platform specification.

Java 2 security can be enabled on the Global Security panel under the WebSphere Administration Console, by enabling the checkbox. For more information on Java 2 Security configuration refer to Section 10.2, "WebSphere Global Security" on page 235.

As a default, when Global security is enabled for WebSphere the Java 2 security is disabled.

WebSphere also maintains a set of policy files for the application server runtime. These files are listed below:

- ► *<WebSphere_root_directory>*\java\jre\lib\security\java.policy
- ► *<WebSphere_root_directory>*\properties\
  - – was.policy
  - – client.policy
  - – server.policy
- ► *<WebSphere_root_directory>*\config\cells\<your_cell>\filter.policy
- ► *<WebSphere_root_directory>*\config\cells*<your_cell>*\nodes*<your_node>*
  - – app.policy
  - – library.policy
  - – spi.policy
- ► was.policy for the applications under each installed application in the META-INF directory.

8.6 JAAS

JAAS (Java Authentication and Authorization Services) is a standard extension to the Java 2 SDK v1.3 and it is part of Java 2 SDK v1.4. The current version for JAAS is 1.0. The WebSphere Application Server V5 also implements and uses JAAS for security purposes.

The best way to learn JAAS is to start with the sample application that comes with JAAS v1.0; download the extension fron Sun's Java site: http://java.sun.com/products/jaas.

8.6.1 Implementing security with JAAS

This section will explain how JAAS is generally used to implement security in Java.

> **Note:** JAAS does not require Java 2 security to be enabled. JAAS can be configured and used without Java 2 security.

Secured application
Talking about security, here we will investigate how security works within an application. The secured application has two parts:

► The main application that handles the login procedure and runs the secured code under the authenticated subject.

► The action that is invoked from the main application under a specific subject.

Supporting Components

The following objects are required for a secured application using JAAS:

► *Principal* that is part of a *Subject*.

First of all, the *Subject* has to be explained. In JAAS the *Subject* is some identity. This identity will be authenticated and permissions will be assigned to it.

A *Subject* can have relationship with several different authorities. In JAAS these multiple interactions with authorities are represented by objects that are implmenting the *java.security.Principal* interface. The principal objects are listed under a subject.

► A callback handler that implements the *CallbackHandler* interface. This class is a client provided set of interfaces for entering authentication information. These interfaces decouple the service provider from the particular input devices being used.

> **Definition:** What is callback ?
>
> Developers conversant in event-driven programming model of Microsoft Windows and X Window are accustomed to passing function pointers that are invoked when something happens. The invocation part of the process is the "callback". However Java does not support method pointers, Java interfaces provide a solution to implement callbacks.

The following callbacks are provided for user interaction:

– ChoiceCallback: collects choice information

– ConfirmationCallback: collects confirmation information

– LanguageCallback: collects the language information

– NameCallback: collects the user name for login

– PasswordCallback: collects the password for the login

– TextInputCallback: collects simple text information

– TextOutputCallback: provides text information

– WSRealmNameCallbackImpl: collects the realm for the login. It is an IBM proprieary callback type.

- WSCredTokenCallbackImpl: collects the token for the login. It is an IBM proprieary callback type.

► A class that implements the *LoginModule* interface, which does the login process effectively.

Three descriptor (configuration) files are also required for JAAS:

► The subject-based access control policy for the application, passed along with the *-Djava.security.auth.policy=* parameter to the JVM.

► The access control policy file for the application, passed along with the *-Djava.security.policy=* parameter to the JVM.

► Login configuration for the application, passed along with the *-Djava.security.auth.login.config=* parameter to the JVM.

8.6.2 How is JAAS security working ?

All you need to do is to start the client in the virtual machine with the correct configuration.

The following system properties configure JAAS for the Java Virtual Machine.

► *-Djava.security.auth.policy=jaas.policy* defines the JAAS policy for the virtual machine

► *-Djava.security.auth.login=login.conf* provides the configuration file for LoginContext, what login class to use for authentication.

Running the application with specific JAAS configuration look like the following example:

```
java -Djava.security.auth.policy=jaas.policy
-Djava.security.auth.login=login.config com.mycompany.MyApplication
```

JAAS requires a configuration file for the login module definition for example: `login.config`.

Example 8-5   login.config

```
WSLogin {
    com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProxy required
    delegate=com.ibm.ws.security.common.auth.module.WSLoginModuleImpl;
};
```

The configuration file can have multiple definitions, each definition has an alias name, that can be used in the application to set the login module. The entry defines the Java class for the login module; the "required" means that this class's approval is necessary for login to succeed. The entry can optionally define a delegation class for the login module.

There can be a policy file for JAAS defined, similar to the policy files used in Java 2 security. Actually, the Java 2 Security and JAAS policy files can be merged in the next release of Java, in J2EE V1.4. This policy file is optional.

Example 8-6   jaas.policy

```
grant Principal SamplePrincipal "user01" {
   permission java.util.PropertyPermission
   "user.home", "read";
};
```

The policy file defines the access policy for the resources in an application. The policy defines the principal for the resources.

8.7 Programmatic login

When you want to implement your own login mechanism for your application you might want to use JAAS and implement the required programmatic login.

The following diagram,Figure 8-8, shows the activity diagram for JAAS how the different components are working together during a login process.
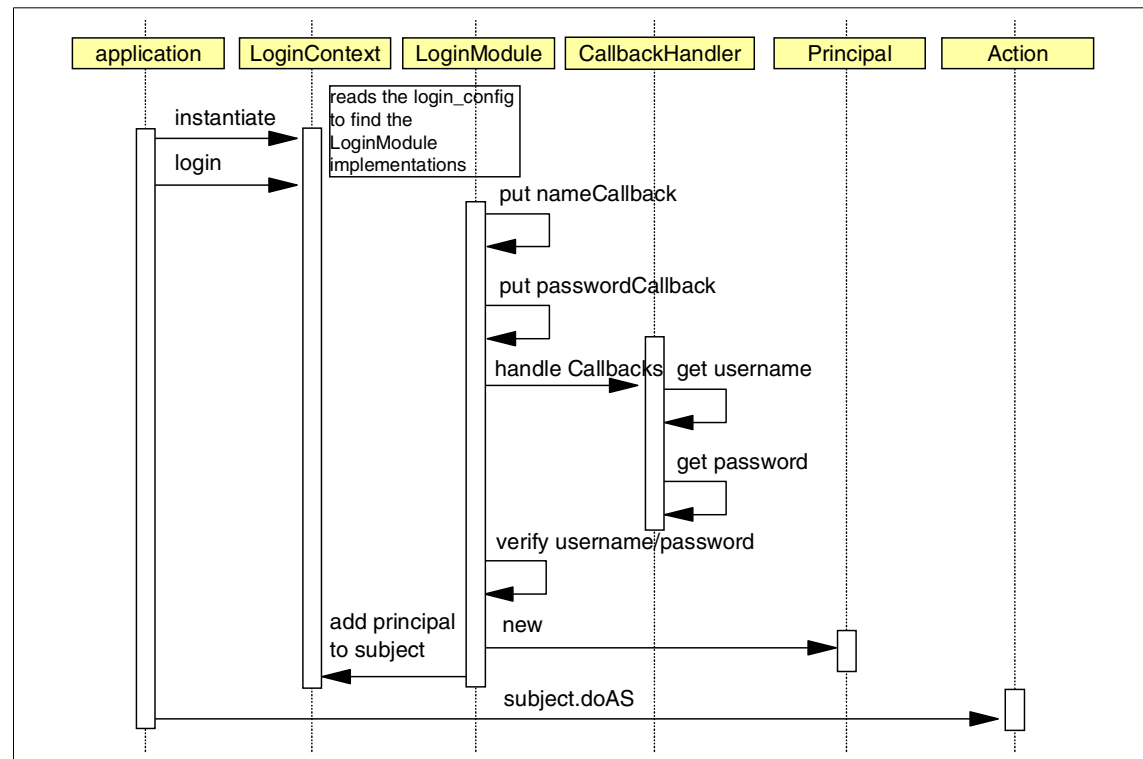
Figure 8-8   JAAS sequence diagram

The step-by-step process is described below:

1. The application starts the login process using JAAS.

2. The *LoginContext* gets initialized.

3. During the login process, executed in the *LoginContext*, a *Principal* will be authenticated using the specified callback handler.

4. If the authentication was successful the *LoginContext* commits the login, the *Principal* gets assigned to the *Subject*.

5. The application gets the *Subject* from the *LoginContext*.

6. The *doAs* method attempts a secured operation under the acquired *Subject*.

8.7.1 JAAS in WebSphere

In the previous version of WebSphere, WebSphere Application Server V4.0, other mechanisms were used to perform programmatic login. There was a distinction between server-side and client-side programmatic login. One was using the LoginHelper class together with CORBA authentication methods, while the other was utilizing the ServerSideAuthenticator class. In WebSphere V5.0 these classes together with the mechanism are deprecated; use the JAAS programmatic login instead.

With JAAS the client-side and server-side login works the same way, however the challenge for authentication works a bit different. On the client-side any challenge mechanism can bu used that is compatible with the client's runtime environment; while on the server side there is no place to pull up an authentication challenge window or provide a command line prompt for username, password. On the server-side, credentials have to be collected in the code then provided to the JAAS login facility.

8.7.2 Client-side login with JAAS

Client-side login is useful when the user needs to login into the security domain on a remote system using the client application. In this case the client application has to collect the login information for authentication purposes. WebSphere provides built-in mechanisms to collect the necessary information: user name, password, realm.

There are two scenarios in this client-side login section, one is an example for J2EE Java applications, and the other is for thin Java applications. For more informaiton about Java client security refer to Chapter 6, "Securing Java clients" on page 97.

J2EE Java application

The following code snippet shows how to perfrom various types of login using character-based console (stdin), graphical user interface, and direct login without login prompt.

Example 8-7   Client side login in the ITSOBank J2EE client

```
...
private static void loginClient() {
   LoginContext lc = null;
   try {
      // using the console (stdin) to collect the login information
      if(logintype.equals("stdin")) {
         System.out.println("Performing stdin login...");
```

```
            lc=new LoginContext("ClientContainer", new
WSStdinCallbackHandlerImpl() );
        }
        // using the graphical interface to collect the login information
        if(logintype.equals("gui")) {
            System.out.println("Performing GUI login...");
            lc=new LoginContext("ClientContainer", new WSGUICallbackHandlerImpl()
);
        }
        // collecting the login information from command line and
        // login directly without prompting for user name, password and realm
        if(lc==null) {
            System.out.println("Performing silent login...");
            lc=new LoginContext("ClientContainer", new
WSCallbackHandlerImpl(loginuid,loginrealm,loginpwd));
        }
    // exception handling
    } catch (LoginException le) {
        System.out.println("Cannot create LoginContext. " + le.getMessage());
        // insert error processing code
        return;
    } catch (SecurityException se) {
        System.out.println("Cannot create LoginContext." + se.getMessage());
        se.printStackTrace(System.out);
        // Insert error processing
        return;
    }
    // performin login based on the login context
    try {
        lc.login();
    // exception handling
    } catch (LoginException le) {
        System.out.println("Fails to create Subject. " + le.getMessage());
        le.printStackTrace(System.out);
        return;
    }
}
...
```

Beside the J2EE container the client requires a properties file for the JAAS login
configuration; a configuration file for CSIv2 and IBM SAS configuration; keystore
files for SSL communicaiton.

Running the client-side login sample

To test the client-side login scenario, launch the ITSOBank J2EE client application from the command line. The client is packaged with all the application modules in the enterprise archive. Normally you will only need the client application, the utility JARs, and some of the EJB classes for client access.

```
launchclient itsobank_secured.ear [ stdin | gui | username password realm ]
```

The following example launches the client using the character-based console to collect login information:

```
launchclient itsobank_secured.ear stdin
```

The following example launches the client passing the login information as parameters:

```
launchclient itsobank_secured.ear manager01 password dirsrv:389
```

If you are running the client from a remote terminal, you have to specify the server hostname and eventually the server port for the connection:

```
launchclient -CCBootstrapHost=appsrv01 -CCBootstrapPort=2809
itsobank_secured.ear
```

Thin Java application

The thin Java application also has to use a login mechanism to log in to the application server. WebSphere Application Server V5 supports JAAS as the authentication mechanism for programmatic login. In order to perform a log in to access a remote EJB you have to do the followings:

1. Initialize the ORB for the CORBA connection. The following code snippet shows the ORB initialization.

Example 8-8   ORB initialization

```
...
Properties props = new Properties();
props.put("org.omg.CORBA.ORBClass", "com.ibm.CORBA.iiop.ORB");
props.put("com.ibm.CORBA.ORBInitRef.NameService","corbaloc:iiop:" + serverName
+ ":" + serverPort + "/NameService");
props.put("com.ibm.CORBA.ORBInitRef.NameServiceServerRoot","corbaloc:iiop:"+
serverName + ":" + serverPort + "/NameServiceServerRoot");
ORB _orb = ORB.init((String[]) null, props);
...
```

2. Perform the programmatic login using JAAS; this part is very similar to the login process introduced before.

3. Initialize the Context for the EJB lookup, then lookup the EJB.

4. Acquire the remote object through the EJB's home interface.

> **Note:** Without initializing the ORB, JAAS will not be able to perform the login. The application will perform the authentication challenge as it is set in the SAS client configuration file, it is the GUI login panel by default.

This sample application uses a callback handler called: *ITSOBankCallbackHandler* implementing the *CallbackHandler* interface. It is a simple implementation collecting the login information from a character-based console. It is only provided to show how to implement a callback handler and use it with JAAS. For more information about the handler, see the comments in the source of ThinAccountViewer, it is at the end of the file.

Running the client-side login sample

To test the client-side login scenario, launch the ITSOBank thin Java client application from the command line. The client is provided together with the ITSOBank sample application as a separate package.

The following example launches the client using the character-based console to collect login information:

```
runclient appsrv01 2809 login
```

The application will collect the login information: user name, password and realm on the character-based console; for example: `manager01`, `password`, `dirsrv:389`.

After a successful authentication the client application GUI comes up, where you can collect balance information for customers and branches.

8.7.3 Server-side login with JAAS

Server-side login is used when the application has to log the users into the security domain by providing authenticaiton data and login information on the server side. In these situations a server-side component, for example: servlet, EJB, performs authentication for the application.

For authentication purposes the Java Authenticaiton and Authorization Services (JAAS) is used on the server-side. Similar to the client-side login the login is performed programatically coded in the component. The user details can be collected in any format and have to be presented through the login context during the login process. The main difference between server-side and client-side login is that on the server-side it is not possible to collect the login information through user interaction. It is not possible to pull-up a graphical window or character-based console to ask for user name and password. The login information (user name, password, realm) is passed directly to the login context.

The following example is a code snippet from the ITSOBank application, performing server-side login for the TransferServlet servlet. Note, that the callback handler at this time is the *WSCallbackHandlerImpl* class, and the login information is passed to the handler as parameters.

Example 8-9   Server-side login in the ITSOBank TransferServlet servlet

```
...
try {
    LoginContext lc=new LoginContext("WSLogin",new
WSCallbackHandlerImpl(loginusername,loginrealm,loginpassword));
    lc.login();
} catch (LoginException le) {
    // handling the exception
}
...
```

The other difference with server-side login is that the login.properties defined under the <WebSphere_root>/properties/ directory is used to configure JAAS.

Running the server-side login sample

In order to test the server-side login function in the ITSOBank sample application, launch a Web browser and access the http://localhost/itsobank application.

Select the **Modified Customer Transfer 2. - using Server-Side Login** link. Fill out the next form with the required information, do not forget to provide the realm, for example: dirsrv01:389 if you are using an LDAP directory on host: dirsrv01 at port: 389.

Submit the transfer, then wait for the response page. When the transfer is done, go and check out the SystemOut.log file under the <WebSphere_root>/logs/server1 directory. Go to the end of the file and look for the identities who initiatied the transfer and who invoked the bean methods.

8.8 Where to find more information

For more information on the security aspects of J2EE, see the following documents:

► The Java 2 Platform Specification v1.3 at
  http://java.sun.com/j2ee/docs.html

► The specification for Java Authentication and Authorization Service (JAAS) and Java 2 Security is also available at the previous URL.

9

WebSphere Application Server Security

This chapter is an introduction, presenting the big picture of the WebSphere Application Server V5 security concepts.

It covers the security architecture and the basic security settings for the server, and can serve as a quick overview of all the security features and services within WebSphere. Details on different security subjects are presented in the appropriate chapters.

9.1 WebSphere security model

The IBM WebSphere Application Server V5 is a J2EE 1.3 compliant Java application server; it implements the required security services as they are specified. The security components are essential parts of the application server architecture. The following description will give a high-level overview of these

9.1.1 WebSphere security in operating environment

Thought it is not the subject of that book, when discussing security of enterprise application run under WebSphere Application Server we should first take a closer look at the environment under which the server will run. This may have a strong influence on the WebSphere security configuration (that is stored in file system) and the overall application runtime security environment.
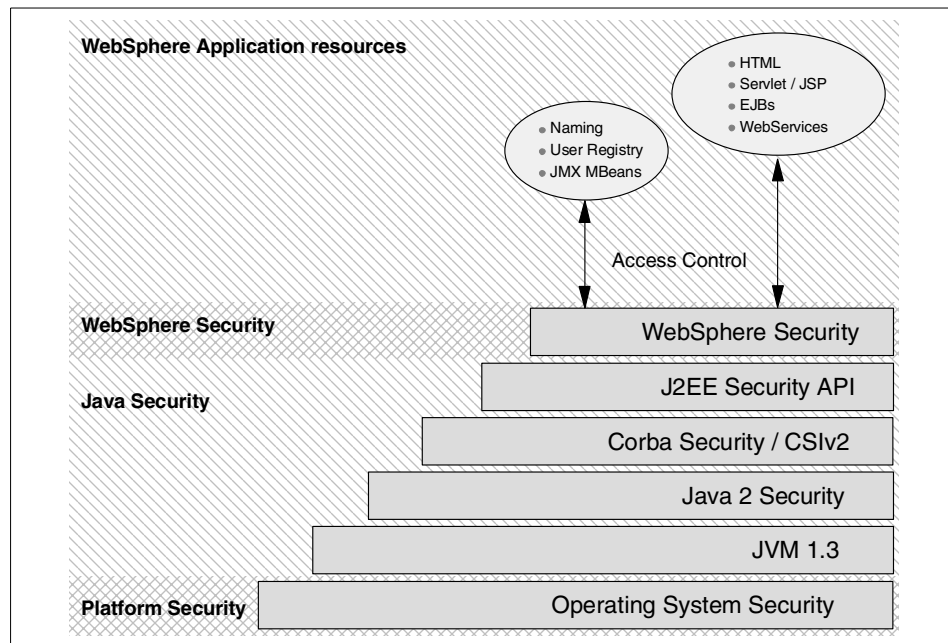
Figure 9-1   WebSphere environment security layers

IBM WebSphere Application Server security sits on top of the operating system security and the security features provided by other components, including the Java language, as shown in figure above.

► Operating system security should be considered in order to protect sensitive WebSphere configuration files and to authenticate users when operating system user registry is used for authentication. This is extremely important in

a distributed WebSphere environment when potentially different operating systems and different user registries might be involved. Keeping the users (and their passwords) and groups in sync across many different machines might be a problematic administration task.

► Standard Java language security is provided through the Java Virtual Machine (JVM) used by WebSphere and the Java security classes.

► Java 2 security enhances standard Java Virtual Machine security by introducing fine grained access, easily configurable security policy, extensible access control structure and security checks for all Java programs (including applets).

► Common Secure Interoperability protocol adds additional security features that enable interoperable authentication, delegation and privileges in CORBA environment. It support interoperability with EJB 2.0 specification and can be used with SSL.

► J2EE security uses the security collaborator to enforce J2EE-based security policies and support J2EE security APIs. APIs are accessed from WebSphere applications in order to access security mechanisms and implement security policies.

WebSphere Application Server V5 security relies and enhances all the above mentioned layers. It implements security policy for in unified manner for both Web and EJB resources.

9.1.2 WebSphere security in distributed environment

In a distributed environment Websphere Application Server may be installed in the Network Deployment configuration, where WebSphere creates a network of application servers instances supporting clustering, caching and efficient utilization of shared resources. Picture below presents a general architecture of the WebSphere deployment configuration.
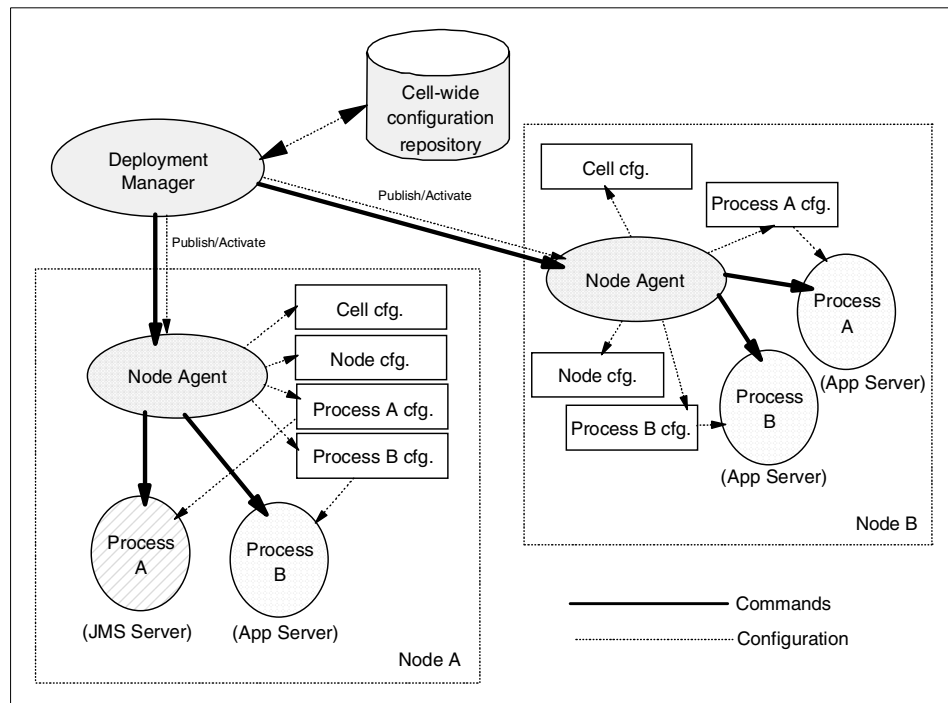
Figure 9-2   Overview of Websphere Network Deployment configuration

Follow the description for the elements on the diagram above:

▶ **Node** is a logical grouping of application server processes. A node often corresponds to physical server machine with IP address assigned to it. Application server processes located on one node are managed by a single node agent. On each mode there is local copy of cell configuration repository managed by a node agent. This repository may be modified through deployment manager during publish/activate configuration processes. Node agent has also access to the configuration repository of each process running on a node.

▶ **Cell** is logical configuration concept that associates WebSphere server nodes with one another. Administrators may freely define the cell according to whatever criteria they will take to group the servers (organizational aspects, application aspects, and so on). A cell is managed by one deployment manager process.

▶ **Node Agent** is administrative process that manages application server processes on a single node. Node Agent routes administrative requests issued from deployment manager to a particular application server. It is purely

administration process participating in the network deployment configuration and is not involved in serving applications.

► **Deployment manager** is an administrative process that controls processes and manges load balancing between the nodes connected to the cell. Administrative access to any node in a cell is governed by deployment manager processes. This means that deployment manager hosts administrative console for entire cell.

In the WebSphere Network Deployment configuration one computer is designated to be central Deployment Manager. This central machine controls other systems that work under its supervision.

Figure 9-3 below presents architectural building blocks of cell in a network deployment manager configuration.
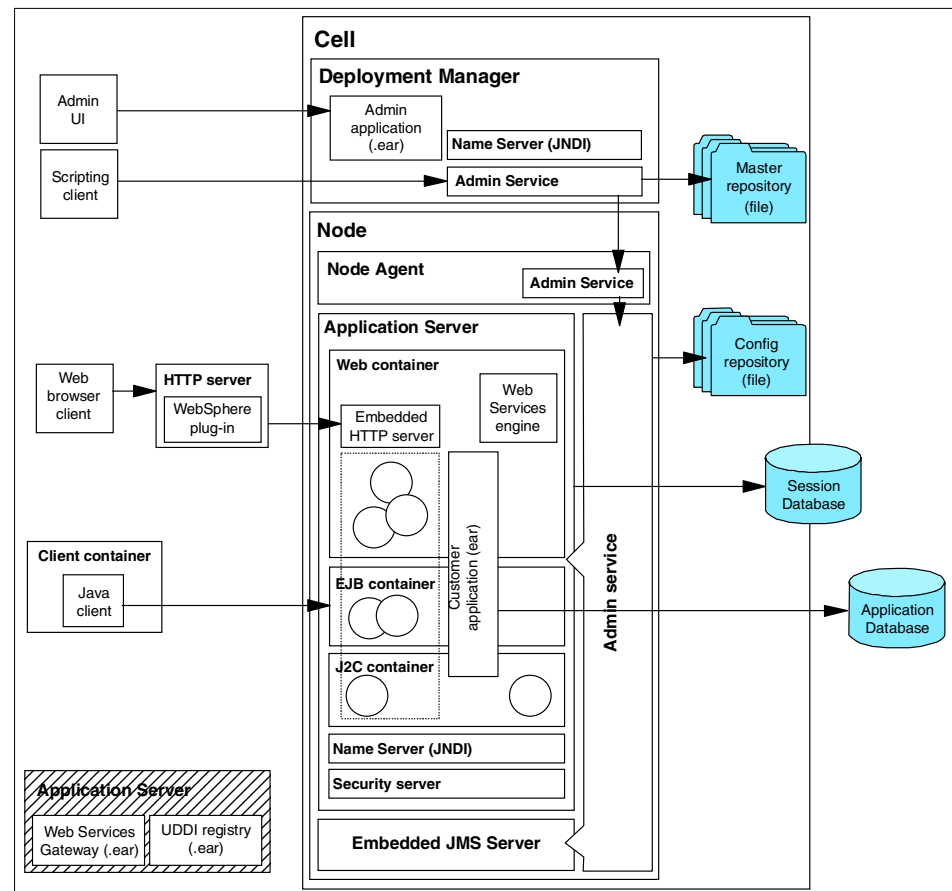
Figure 9-3   Architectural building blocks for WebSphere Network Deployment

From the security point of view each application server process may be configured to have a local or a remote Security Server. This notion of local or remote is relative to the application server process. Deployment Managers and Node Agent processes always have a Security Server and are used to perform authentication and authorization for the Deployment Manager, Node Agent, and for those application servers that were configured to use a remote Security Server.

On some operating systems, like unix for example, running security server will require the highest user priority. This is because of the access to the operating system functions in order to perform authentication based on local operating system user registry requires root privilege.

Many customers will not want to run application server processes with the root privilege because this implies that all application components (like servlets) will have this privilege. In such cases you have to consider how to configure security and when to place security server before deploying an application.

9.1.3 Java Management Extension Architecture (JMX)

Java Management Extension is a set of new interfaces and Java Beans that allows to perform configuration and management of WebSphere Application Server components from custom applications without using the Administrative Console or the wsadmin program.

JMX uses Tivoli implementation scheme and provides client interface and sripting facility that allows the use of MBeans to manage Websphere Application Server. It is used for overall WebSphere system management tasks.

Distributed administrative processes that run on each node have separated administrative repository. Deployment manager that controls a group of nodes provides the scope of visibility of the administrative processes in cell. This is done through node managers as described in the section above. Below is a conceptual diagram of how JMX MBeans are used to perform different administrative tasks.
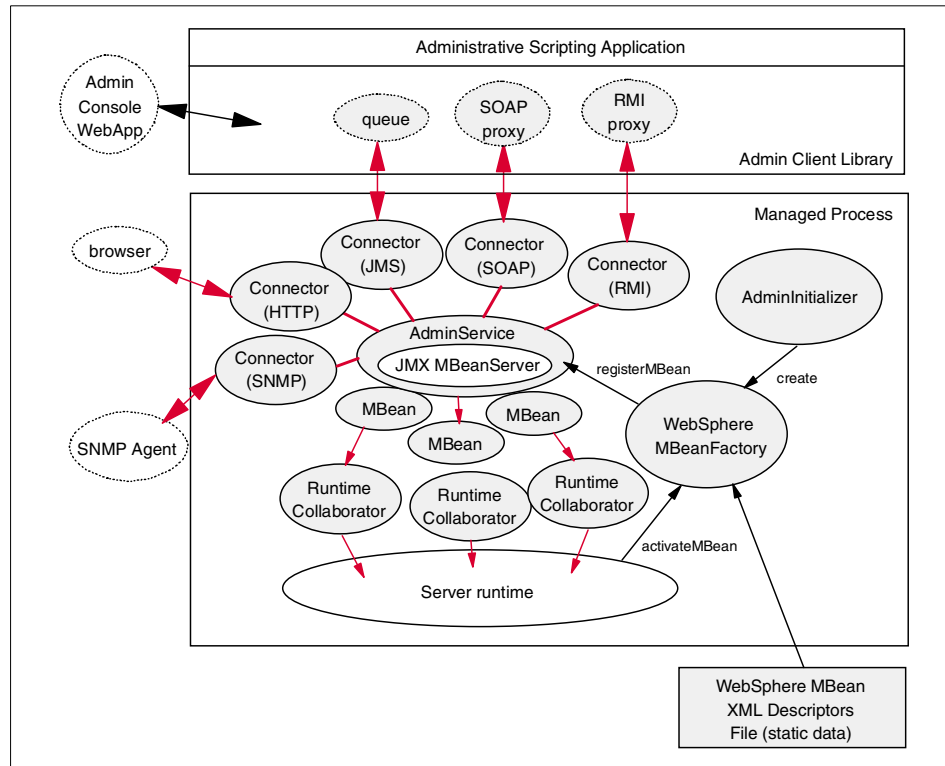
Figure 9-4   JMX Beans in a system management concepts

9.2 Websphere Application Server security architecture

WebSphere Application Server v5.0 is a J2EE 1.3 compliant Java application server, it uses declarative security model in which application expresses security constraints in a form that is external to the application. This external form of the constraint allows the application to be independent on the chosen security mechanism. Security mechanisms are defined and configured as a part of the global application server security.

► **Global Security** specifies global security configuration for a managed domain and applies to all applications running on WebSphere Application Server. It determines whether security will be applied at all, sets up the user registry against which the authentication will take place, defines authentication mechanisms and so on. Global security is managed from the Administrative Console. You will notice that many of the values for global security act as defaults.

▶ **Application Security** determines application specific requirements. In some cases these values may override global security settings but in most cases they complement it. Application security includes such elements as: method for authenticating the users, mechanism for authorizing the users into application specific resources, roles based access control to these resources, roles to user/user groups mapping and so on. Application security is administered during assembly phase using application assembly tool (AAT) and during deployment phase using WebSphere administrative console and wsadmin client program.

Diagram below presents a general overview of building blocks that of single application server and how they interact in a Java client and Web browser communication. For more detailed description of the security communication flow please refer to section , "Other security components" on page 226.
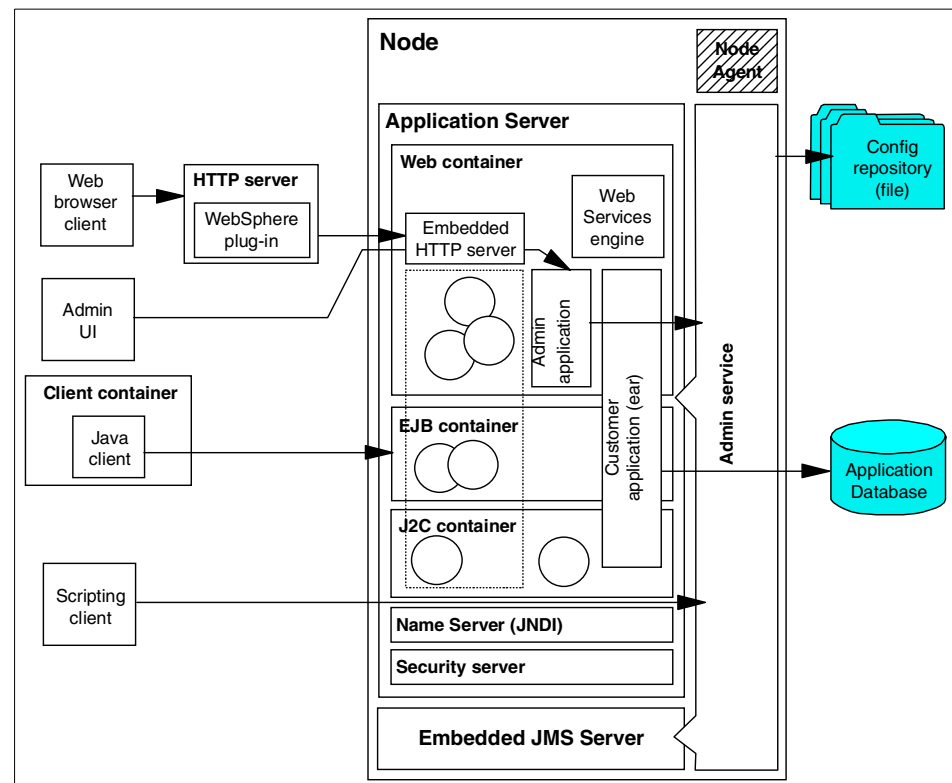
Figure 9-5   Single Application Server building blocks

This section presents general overview of security components of the Websphere Application Server and how they are used to create flexible pluggable architecture.

9.2.1 Extensible security architecture model

A diagram below presents general view of the logical layered security architecture model of WebSphere Application Server 5.0.

Flexibility of that architecture model lies in pluggable modules that can be configured according to the requirements and existing IT resources.

Interface layer allows to connect different modules responsible for authentication, authorization and user registry.

Pluggable user registry allows to configure different databases to store user IDs and passwords that are used for authentication. Detailed information on how to interface to custom registry using *UserRegistry* interface please refer to Chapter 8, "Programmatic security" on page 179.
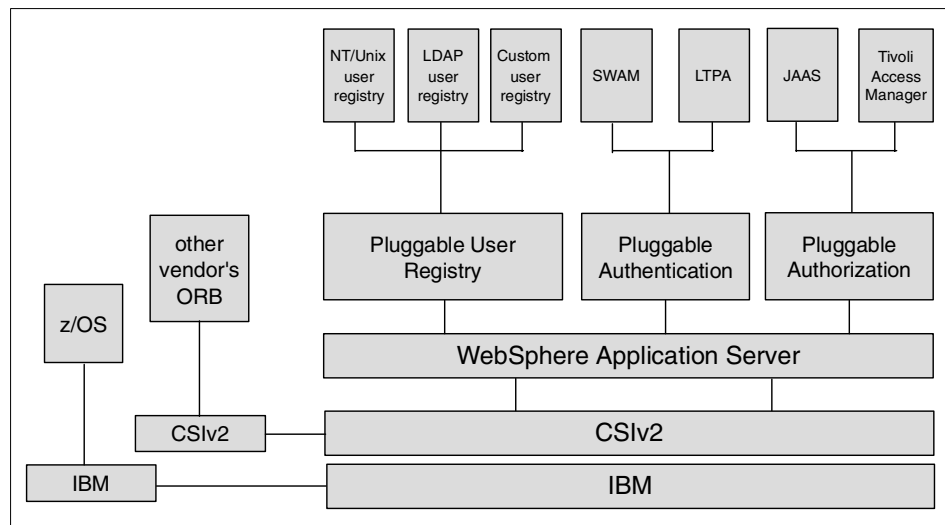
Figure 9-6   WebSphere V5 extensible security architecture

Pluggable authentication module allows to chose whether WebSphere will authenticate the user or will accept the credentials from external authentication mechanisms. On how to configure WebSphere to use credentials from IBM Tivoli Access Manager please refer to Chapter 12, "Tivoli Access Manager" on page 367. In the future this authentication interface will be extended to include other external authentication systems.

Pluggable authorization interfaces will allow to use different authorization mechanisms for WebSphere applications. In a current version, JAAS is supported and Tivoli Access Manager as an external authorization system.

9.2.2 WebSphere Application Server security components

Find the WebSphere Application Server security components listed below.

User registry

User registry stores user and group names for authentication and authorization purpose. Authentication mechanisms configured for WebSphere Application Server consults the user registry to collect user related information when creating credentials which are then used to represent the user for authorization. The options for user registries to set up include:

► **Local operating system user registry** - when configured WebSphere uses the operating system's users and groups for authentication. When configuring Websphere Application Server in Windows NT or Windows 200 platforms that are connected to a Windows domain you should be aware that domain user registry takes precedence over local machine's user registry.

► **LDAP user registry** - in many solutions LDAP user registry is recommended as the best solution for large scale Web implementations. Most of the LDAP servers available on the market are well equipped with security mechanisms that can be used to securely communicate with WebSphere Application Server. WebSphere supports a few LDAP server: IBM SecureWay Directory, Netscape LDAP Server, Lotus Domino LDAP Server, Microsoft Active Directory. There is also a possibility to use other LDAP servers. Flexibility of search parameters that administrator can set up to adapt WebSphere to different LDAP schemas.

► **Custom user registry** - this leaves open door for any custom implementation of user registry database. Websphere API provides the *UserRegistry* Java interface that you should use to write custom registry. This interface may be used to access virtually any relational database, flat files and so on.

WebSphere authentication mechanism can not be configured to use more than one user registries at a time. Only one single active registry is supported and it is set up when configuring Global Security settings using the Administration Console.

Authentication mechanisms

An authentication mechanism defines rules about security information, for example, whether a credential is forwardable to another Java process, and the format of how security information is stored in both credentials and tokens.

Authentication is the process of establishing whether a client is valid in a particular context. A client can be either an end user, a machine, or an application.

An authentication mechanism in WebSphere typically collaborates closely with a User Registry. The User Registry is the user and groups accounts repository that the authentication mechanism consults with when performing authentication. The authentication mechanism is responsible for creating a credential which is a WebSphere internal representation of a successfully authenticated client user. Not all credentials are created equal. The abilities of the credential are determined by the configured authentication mechanism.

Although WebSphere provides several authentication mechanisms, only a single "active" authentication mechanism can be configured at once. The active authentication mechanism is selected when configuring WebSphere global security.

WebSphere provides two authentication mechanisms; Simple WebSphere Authentication Mechanism (SWAM) and Lightweight Third Party Authentication (LTPA). These two authentication mechanisms differ primarily in the distributed security features each supports.

- ► **SWAM** (Simple WebSphere Authentication Mechanism)

    The SWAM authentication mechanism is intended for simple, non-distributed, single application server type runtime environments. The single application server restriction is due to the fact that SWAM does not support *forwardable* credentials. What this means is that if a servlet or EJB in application server process 1, invokes a remote method on an EJB living in another application server process 2, the identity of the caller identity in process 1 is not transmitted to server process 2. What is transmitted is an unauthenticated credential, which, depending on the security permissions configured on the EJB methods, may cause authorization failures.

    Since SWAM is intended for a single application server process, single-sign-on (SSO) is not supported.

    The SWAM authentication mechanism is suitable for simple environments, software development environments, or other environments that do not require a distributed security solution.

    SWAM relies on the session id, it is not as secure as LTPA, therefore using SSL with SWAM is strongly recommended.

- ► **LTPA** (Light Weight Third Party Authentication)

    Lightweight Third Party Authentication (LTPA) is intended for distributed, multiple application servers and machine environments. It supports forwardable credentials and SSO. LTPA is able to support security in a distributed environment through the use of cryptography. This permits LTPA to encrypt and digitally sign and securely transmit authentication related data and later decrypt and verify the signature.

LTPA requires that the configured User Registry is a central shared repository such as LDAP or a Windows Domain type registry.

The following table summarizes the Authentication Mechanism capabilities and user registries used with LTPA.

Table 9-1   Authenitcation mechanisms

AuthC mech.	Forwardable user credentials	SSO	Local OS user registry	LDAP user registry	Custom user registry
SWAM	no	no	yes	yes	yes
LTPA	yes	yes	yes	yes	no

Future versions of WebSphere will support the Kerberos authentication mechanism to provide broader selection and an industry standard mechanism for authentication.

Authorization mechanisms

Websphere Application Server standard authorization mechanisms are based on J2EE security specification and Java Authentication and Authorization Services. JAAS extends the security architecture of the Java 2 Platform with additional support to authenticate and enforce access controls upon users.

JAAS programming models permits the developer to design application authentication in a pluggable fashion, which makes the application independent from the underlying authentication technology.

Java 2 security architecture uses security policy to specify who is allowed to execute a code of the application. Code characteristics, like code signature, signer id, source server, decide whether the code will be granted access to be executed or not. JAAS extends this approach with role based access control. Permissions to execute a code is granted not only on the code characteristics but also on the user, who is running it.

For each authenticated user a Subject class is created and set of Principals are included in the subject in order to identify that user. Security policies are granted based on possessed principals.

Other security components

The list below will show you other security components within the WebSphere Application Server V5.

Security server

Security server is a component of Websphere Application Server that runs in each application server process. If multiple application server instances are executed on a single node, then multiple security servers exist on that node

Security Server component is responsible for managing authentication and authorization, it collaborates with authorization engine and User Registry.

Security collaborators

Security collaborators are application server processes responsible for enforcing security constraints specified in deployment descriptors. They communicate with security server every time when authentication and authorization actions are required. The following security collaborators are identified:

► **Web security collaborator** residing in the Web container.

   Web collaborator provides the following services to the application:

   – checks authentication

   – performs authorization according to the constraint specified in deployment descriptor

   – logs security tracing information

► **EJB security collaborator** residing in the EJB container.

   EJB collaborator uses CSIv2 and SAS to authenticate Java client requests to enterprise beans. It works with the security server to perform the following functions:

   – check authorizations according to specified security constraint

   – support communication with local user registry

   – log security tracing information

   – communicates external ORB using CSIv2 when request for a remote bean is issued

JMX MBeans

Java Management Extension Beans is used in Websphere Application Server V5.0 for management and administration related tasks. Please refer to 9.1.3, "Java Management Extension Architecture (JMX)" on page 220 for more detailed architecture view.

Example below briefly presents how these components interact in three different communication scenarios.
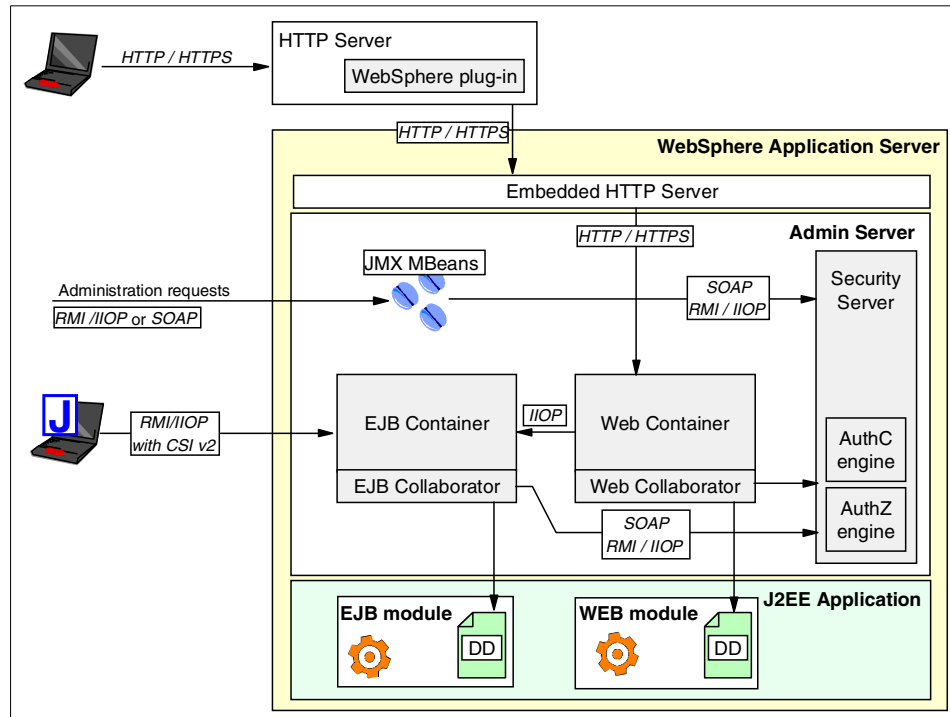
Figure 9-7   Websphere Application Server security components communication

Web browser communication

The steps below describe the interaction of the components from a security point of view when a Web browser sends a request to a WebSphere application.

1. Web users request a Web resource protected by WebSphere application server.

2. Web server receives the request and recognizes that requested resource is on the application server and using WebSphere plug-in redirects the request.

3. Authentication occurs depending on the authentication method selected for the application. WebSphere plug-in passes the user credentials to the Web collaborator which performs user authentication

4. After successful authentication original Web request reaches Web container, which uses Web collaborator to communicate with Security Server for Authorization.

5. Web collaborator passes user's credentials and security information read from deployment descriptor to the security server and gets the response whether access to specified resource is allowed or denied.

6. On subsequent request only authorizations checks are performed either by Web collaborator or EJB collaborator depending on what the user is requesting. Users credentials are extracted from established security context.

Administrative tasks

The steps below follow the process how the administration tasks are exectuted.

1. Administrative tasks are issued using either the Web based Administrative Console or the wsadmin scripting tool.

2. Administration client generates a request that reaches server side ORB and JMX MBeans, JMX MBeans represent managed resources and are part of the management interface system for components. Default communication protocol is SOAP. It can be changed either by giving parameter to the *wsadmin* program or modifying administration settings through the Administrative Console.

3. JMX Beans contact security server for authentication and authorization purpose. JAMX beans have dedicated roles assigned and do not use user registry for authentication and authorization.

Java Client communication

The steps below describe how a Java client interacts with a WebSphere application.

1. Java clients generates a request that reaches server side ORB. Request may got through CSIv2 interceptor if the client's request is generated using CSIv2.

2. Server side ORB passes then the request to EJB container.

3. Authentication is done by the ORB, before client gets access to protected resources.

4. After submitting a request to access protected EJB method EJB container passes the request to the EJB collaborator.

5. EJB collaborator reads deployment descriptor from the .ear file and users credential from security context.

6. Credentials and security information is passed to security server that validates user access right and passes this information back to the collaborator.

7. After receiving a response from Security server EJB collaborator authorizes or denies access for the user to requested resource.

For more detailed information on securing particular J2EE application modules please refer to appropriate sections.

9.3 Performance considerations

From the performance point of view there are few things to consider when designing a secure solution.

The authorization process brings additional load to the application server. In a distributed environment authorization server should be put onto separate machine in order to off load application processing. The following three settings can help to fine-tune the security related configurations to enhance performance.

► Security Cache Timeout

It is set to indicate how long WebSphere should cache information related to permission and security credentials. When the cache timeout expires, all cached information becomes invalid. Subsequent requests for the information result in a database lookup. Sometimes, acquiring the information requires invoking an LDAP-bind or native authentication, both of which are relatively costly operations in terms of performance.

► HTTP Session timeout

This parameter specifies how long a session will be considered active when it is unused. After the timeout the session expires and another session object will need to be created. With high volume Web sites this may influence the performance of the server.

► Registry and database performance

Databases and registries that WebSphere Application Server is using have an influence on WebSphere Application Server performance. Especially in distributed environment when Authorization process uses LDAP server you have to consider tuning LDAP database and LDAP server for performance before starting to tune WebSphere.

9.4 Authentication summary

The following diagram is a summary of the authentication mechanisms for the different kind of clients (clients here is used as the most general term).
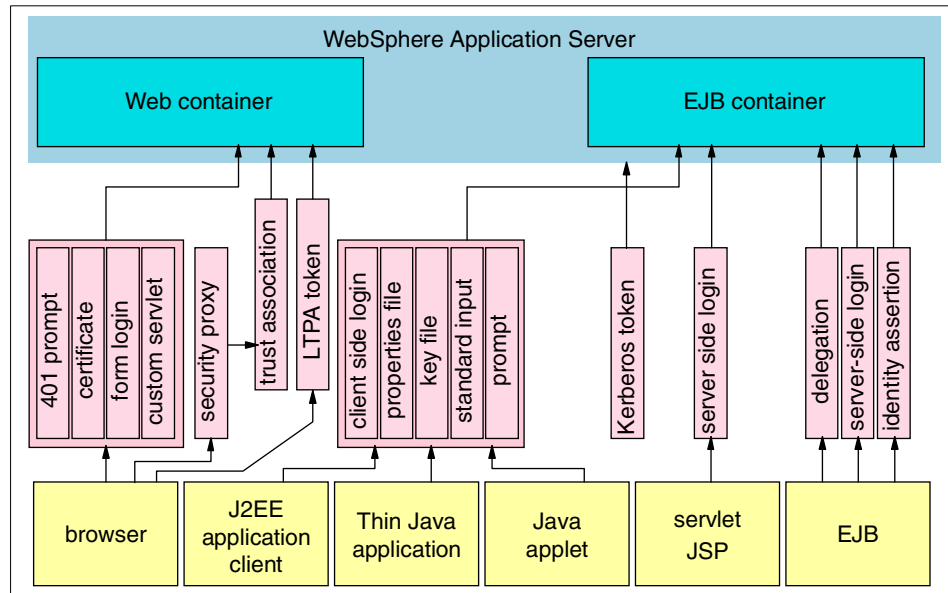
Figure 9-8   Authentication mechanisms overview

At the bottom of this diagram are the different types of clients. These programs can implement EJB clients to access the EJBs.

At the top of the diagram are two containers provided by the application server: the Web container and the EJB container. The clients try to access the assets served by the containers (HTML pages, servlet, JSPs, EJBs).

At the center of the diagram are the authentication mechanisms.

The clients can use authentication mechanisms listed in the center to reach the appropriate container.

10

Administering WebSphere security

This chapter describes in further detail the steps necessary to configure security and secure communication between the various components in WebSphere V5.

The discussion follows the navigation from the Administrative Console to introduce all the security settings and configurations for WebSphere Application Server.

You will find sections in the book referring to sections of this chapter instead of duplicating administrative tasks and configuration aspects.

The topics covered in this chapter essential resource for system adminstrators, system managers and WebSphere administrators.
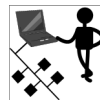
10.1 Administration tools

WebSphere allows administration in many different ways. It provides a good browser based GUI application, a command line utility called wsadmin, and a programmatic API allowing custom Tools development.

► **Administrative Console**

Administrative Console for V5.0 is a browser based GUI application which is more sophisticated, much better, very useful and easy for even remote administration than previous versions of Administrative Consoles. Though remote invocation of Administrative Console of the previous versions Java application was also possible but it was not easy to use because of the slow response, and problematic firewall configurations.

The URL for the Administrative Console is `https://<serverName>:9090/admin`. WebSphere Application Server V5.0 Administrative Console is more secured than the previous versions Administrative Console, now we can define role-based access for Administrative Console, it supports four different roles: Monitor role, Operator role, Administrator role, and Configurator role. When Global Security is enabled, we can define users for the different roles WebSphere allows.

> **Note:** use the fully qualified name of the server when you access the Administrative Console with a Web browser (for example: `wassrv01.itso.ral.ibm.com`). When Single Sign-On is enabled it requires the fully qualified name to identify the domain. If the domain suffix is not specified you will not be able to login to the administrative application.
>
> TEST WITH FINAL RELEASE, WITH AND WITHOUT CONFIGURING DNS

► **wsadmin scripting tool**

WebSphere Application Server V5.0 wsadmin is a command line utility for administration and configuration. wsadmin uses the Bean Scripting Framework(BSF) and supports the JACL scripting language. wsadmin scripts use Java Objects for application management, configuration, operational control, and for communication with MBeans running in WebSphere server process.

Please refer to Appendix D, "Using wsadmin scripting for security configuration" on page 509 for security configuration for some examples for wsadmin scripts.

► **Custom Tools developed by using Programmatic API**

JMX Management API in WebSphere allows the user to create any resource programmatically without using GUI. Using this API you are allowed to write applications that can perform configuration of WebSphere Application Server V5.0.

Please refer to <WebSphere_root>\web\apidocs\index.html for API documentation for JMX Management.

Security trace

To trace the security processes in WebSphere Application Server V5, enable tracing for WebSphere and set the following trace conditon, in addition to your own settings:

```
com.ibm.ws.security.*=all=enable:SASRas=all=enabled:
```

Once the server is restarted with the new settings you will find a trace.log file in your server's log directory.

10.2 WebSphere Global Security

What does WebSphere mean by *security*? A secure application, for instance, will demand certain information is presented before responding to a request for service and given a lack of appropriate information will not be prepared to perform as requested. An application may consist of several parts, or components, and WebSphere implements the means for securing each part. Chapter 4, "Securing Web components" on page 37, Chapter 5, "Securing EJBs" on page 73 and Chapter 6, "Securing Java clients" on page 97 covers how to start securing the common elements of a J2EE application and describes how the information necessary to access a secured service is gathered.

Security is also required during communication between the service requestor, the client, and the service provider, the server. Configuring security, in this sense, can be a complex process. It is necessary to apply appropriate measures at each point of a network and to ensure that they are functioning as required. Every component in an *end-to-end solution*, from client to server must be capable of providing enough security to ensure that information that passes through that point cannot be compromised.

A Web browser, for instance must be capable of sending HTTP requests in a secure fashion since this data may travel over an insecure connection to the server, that is a connection liable to eavesdropping or other interference. The server, upon receiving a request must be able to summon the appropriate resources in order to respond without revealing information unnecessarily to either the resource or a third-party.

As a request passes from one component to another, the opportunities for the interception and exposure of information increases and ultimately the overall security of a system directly relates to the weakest, or least secure, point. WebSphere and indeed J2EE does not implicitly provide a secure means of communication but rather relies on an additional service, typically a transport-layer digital encryption algorithm, called Secure Sockets Layer, (SSL) and Transport Layer Security (TLS). This section describes how to configure WebSphere to use SSL to protect information as it is communicated from the client to the server and back.

Security Center

The Security Center is the focal point for the configuration of WebSphere security. It is accessible from the Admin Console. After logging in, click the Security Center link in the naviagtion pane.

WebSphere security can be enabled and disabled in its entirety by selecting a single switch. This is the Global Security *Enabled* switch which is accessible from the Administrative Console under **Security -> Global Security**.

Figure 10-1   Global Security configuration

Enabling security refers to activating the security settings for the particular Security Server. For a simple overview of the Security Server, refer to Chapter , "Other security components" on page 226.

To enable Global Security, certain criteria must be met.

► An authentication mechanism must be selected. The Application Server supports two authentication mechanisms by default, SWAM and LTPA and SWAM is selected initially. Refer to Section , "Authentication mechanisms" on page 224 for a description of WebSphere's authentication mechanisms. LTPA

requires some additional configuration, which is documented in Section 10.6, "LTPA" on page 249.

► A user registry must be selected. The Application Server supports the concept of a custom registry, which makes the integration of WebSphere with any type of appropriate registry fairly straight-forward. LocalOS and LDAP are the two types of registry provided by default and LocalOS is selected initially. Refer to Section , "User registry" on page 224 for a description of WebSphere's user registries. For information regarding the development of a custom registry, look at Chapter 8.3, "CustomRegistry SPI" on page 183. Additional configuration is required for the user registry, which is documented in Chapter 10.4, "Configuring a user registry" on page 243.

> **Note:** Global Security must be enabled in order for any of the security mechanisms to operate. Disabling Global Security has the effect of turning off all security checks, including checks made when accessing the Admin console.

Other configuration options on the Global Security page are:

► Enforce Java 2 Security. This option is disabled by default, but may be enabled by checking this option. Refer to Chapter 8.5, "Java 2 Security" on page 195 for details regarding Java 2 security managers.

► User Domain Qualified User IDs. If this option is enabled, user names will appear with their fully-qualified domain attribute when retreived programmatically.

► Cache Timeout. When the timeout is reached the Application Server clears the security cache and rebuilds the security data. Since this affects performance, this value should not be set too low.

► Issue Permission Warning. The filter.policy file contains a list of permissions that an application should not have according to the J2EE 1.3 Specification. If an application is installed with a permission specified in this policy file and this option is enabled, a warning will be issued.

► Active Protocol. Determines which ORB-based authentication protocols are accepted by the Application Server. Refer to Chapter 6.2, "CSIv2 and SAS" on page 100 for a description of the CSI specification.

Once Global Security is enabled, user identification must be provided to start and stop WebSphere.

► If the startServer script is started from the WebSphere service, then provide the identity in the server entry.

► If the startServer and stopServer scripts are started from the command line with no additional options, then the identity that the command line shell is

operating under will be used to start the server. Therefore it is necessary to ensure that this identity has the authority to start and stop WebSphere (see below).

► If the startServer and stopServer scripts are started from the command line with *username* and *password* options, then WebSphere will start under the supplied identity. For example:

```
stopserver server1 -username <admin_name> -password <admin_password>
```

It is also necessary to provide appropriate details to log into the Administrative Console. To gain access to the Administrative Console, the user must either

► log in as a user that is a member of one of the four WebSphere administrative roles (see Chapter 10.3, "Administrative roles" on page 239 for details)

► log in under the identity supplied in the user registry panel.
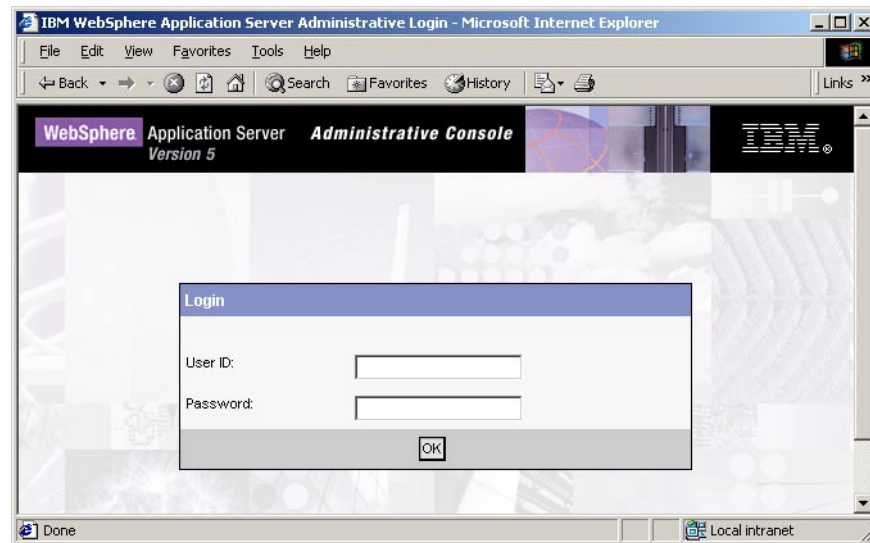
Figure 10-2   The Administrative Console requires a valid administrative identity

10.3 Administrative roles

The J2EE role-based authorization concept has been extended to protect the WebSphere Administrative subsystem. Four roles are defined for performing administrative tasks.

Table 10-1   WebSphere Administrative roles

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

Note: The Admin roles are effective only when Global Security is enabled.

The identity that is specified when enabling Global Security is automatically mapped to the Administrator role. Therefore, it is not necessary to manually add this identity to the administrator role.

Users and groups, as defined by the user registry, 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, it is advisable to map groups to administrative roles so that users may be added to the groups appropriately (and hence the users are mapped to administrative roles) without the need to restart the WebSphere server.

Mapping a user to an administrator role

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

1. From the Admin Console, click **System Administration -> Console Users.**

2. Click **Add.**

3. Enter a user name in the User text box. This user must be defined in the user registry that will be active when Global Security is enabled.

4. Select the appropriate administrative role; more than one role may be selected.

5. Click **OK**. If the user cannot be found in the registry, then a error will occur.

6. Ensure the new mapping is in the Console Users list.

7. Save the change to the master configuration using the link provided at the top of the window and restart the server.

Figure 10-3   Mapping a user to an Administrative role

Mapping a group to an administrator role

As mentioned earlier, it is advisable to map groups to roles rather than users. Mapping a group is similar to mapping users.

1. From the Admin Console, click **System Administration -> Console Groups**

2. Click **Add**.

3. Either a specific group may be mapped or a special subject.

   To map a specific group, enter the group name in the Specify group text box. This group must be defined in the user registry that will be active when Global Security is enabled

   To map a special subject, select the Select from special subject option and select the appropriate subject from the drop-down list. A special subject is a generalization of a particular class of users. The AllAuthenticated special subject means that the access check of the admin role ensures that the user making the request has at least been authenticated. The Everyone special subject means that anyone, authenticated or not, can perform the action, as if no security were enabled.

4. Select the appropriate administrative role; more than one role may be selected.

5. Click **OK**. If the group cannot be found in the registry, then an error will occur.

6. Ensure the new mapping is in the Console Groups list.

7. Save the change to the master configuration using the link provided at the top of the window and restart the server.
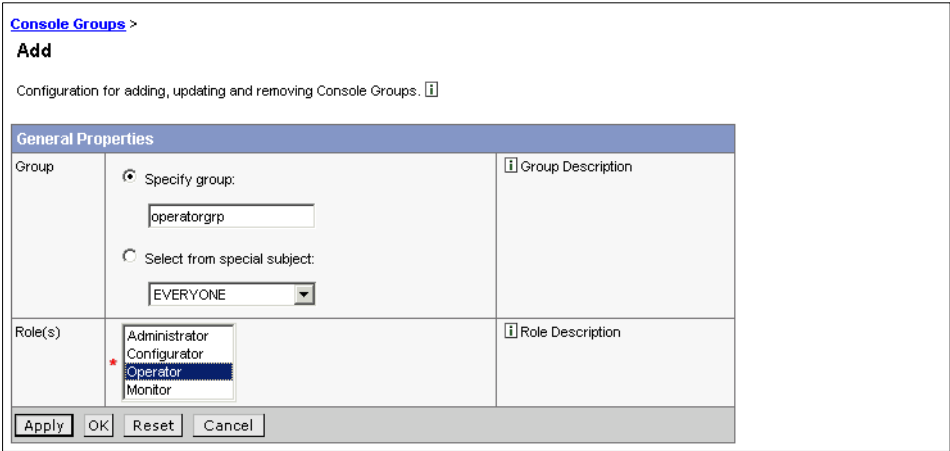
Figure 10-4   Mapping a group to an Administrative role

10.3.1 Cos Naming roles

The J2EE role-based authorization concept has been extended to protect the WebSphere CosNaming service. CosNaming security offers increased granularity of security control over CosNaming functions, which affect the content of the WebSphere Name Space. There are generally two ways in which client programs will make a CosNaming call. The first is through the JNDI interfaces. The second is CORBA clients invoking CosNaming methods directly. Four roles are defined.

Table 10-2   CosNaming roles

Role	Description
Cos Naming Read	Users who have been assigned the CosNamingRead role will be allowed to do queries of the WebSphere Name Space, such as through the JNDI lookup method. The special-subject, Everyone, is the default policy for this role.
Cos Naming Write	Users who have been assigned the CosNamingWrite role will be allowed to do write operations such as JNDI bind, rebind, or unbind, plus CosNamingRead operations. The special-subject, AllAuthenticated, is the default policy for this role.

Role	Description
Cos Naming Create	Users who have been assigned the CosNamingCreate role will be allowed to create new objects in the Name Space through such operations as JNDI createSubcontext, plus CosNamingWrite operations. The special-subject AllAuthenticated is the default policy for this role.
Cos Naming Delete	Users who have been assigned CosNamingDelete role will be able to destroy objects in the Name Space, for example using the JNDI destroySubcontext method, as well as CosNamingCreate operations. The special-subject AllAuthenticated is the default policy for this role.

Note: The CosNaming roles are effective only when Global Security is enabled.

Mapping a user to a CosNaming role

The process of mapping a user to a CosNaming role is the same as mapping users to the admin role. The difference is that you have to navigate to the **Environment -> Naming -> CORBA Naming Service Users** from the Security Center. The roles are also different that you can apply to a user.

Mapping a group to a CosNaming role

The process of mapping a group to a CosNaming role is the same as mapping groups to the admin role. The difference is that you have to navigate to the **Environment -> Naming -> CORBA Naming Service Groups** from the Security Center. The roles are also different that you can apply to a group.

10.4 Configuring a user registry

Configuring a user registry for the application server is tightly related to security, although we have to notice that global security does not have to be enabled in order to use a user registry configured for WebSphere. For example: you can deploy your application and assign groups and users from the actual user registry without security being enabled for the application server.

A brief description covering user registries is provided in Chapter , "User registry" on page 224. For information regarding the development of a custom registry, refer to Chapter , "Developing a custom registry" on page 184.

When WebSphere performs authentication, it sends a request to the appropriate registry. This authentication request may be sent under an identity that differs from that which WebSphere is currently running under. This request identity must be set in the Administrative Console as part of the configuration process.

WebSphere Application Server V5.0 provides three types of user registries:

▶ Local OS User Registry

▶ LDAP User Registry

▶ Custom User Registry

10.4.1 LocalOS

To configure WebSphere to use the local Operating System's registry, open the Admin Console and select **Security -> User Registries -> Local OS**.

1. Enter a user name and password. This is the identity under which the request is sent to the registry. The LocalOS registry requires an identity for a user that has administrative capabilities, as defined by the LocalOS registry.

2. Click **OK**. (It should not be necessary to set any custom properties for the LocalOS registry).

Figure 10-5   LocalOS registry user name and password

3. If there are no errors at this stage, select **Security -> Global Security**. Ensure that the Active User Registry option is set to **LocalOS** and that Global

Security is **enabled**. If this is not the case, make the necessary changes and then click **OK.**

Active User Registry	Local OS ▼	ⓘ Specifies the active User Registry when security is enabled.
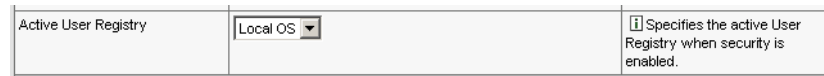
Figure 10-6   LocalOS is set for Global Security

4. The changes will need to be saved and WebSphere restarted.

10.4.2 LDAP

To define WebSphere's LDAP configuration perform the following steps:

1. In the Admin Console, select **Security -> User Registries -> LDAP**.

2. Provide the details for the fileds on the **Configuration panel** as listed below.

   Server Id: This is the WebSphere administrator id. Here we use `cn=wasadmin,o=itso` as the administrator id.

   Server Password: This is the WebSphere administrator password. Specify your password. Specify the same password that is there in the LDAP registry.

   Type: This is the type of LDAP server. Here we are using IBM SecureWay Directory, so select the `SecureWay` from the drop-down list given.

   Host: Specify the hostname of the LDAP server host name.

   Port: This is the port number LDAP server is running on, by default it is 389.

   Base DN: Specify the base DN of your LDAP configuration. In our case we set the Base DN to `o=itso`.

   Bind DN: This is the distinguished name for the application server to use to bind to the LDAP server. In our case it is `cn=root`.

   Bind Password: This is the password for the application server to use to bind to the LDAP server. In our case it is `password`.

   Reuse Connection: Generally this value should be checked. In rare situations when you use a router to spray the requests to multiple LDAP servers and this router does not support affinity, we disable this checkbox.

Figure 10-7   LDAP Settings for WebSphere Application Server

3. Click **Apply**.

4. Save the configuration for WebSphere.

5. We need to define the configuration for Global Security in the Administrative Console.Administrative Console, navigate to **Security -> Global Security**.

6. In the Configuration tab provided, fill up the following values as mentioned below.

7. For **Active User Registry** drop-down list select **LDAP** as the active user registry when security is enabled.

| Active User Registry | LDAP ▼ | ⓘ Specifies the active User Registry when security is enabled. |

Figure 10-8   LDAP is the Active User Registry

8. Click **Apply**, that will validate the settings.

> **Note:** In case the validation fails for any reason, go back to the LDAP configuration panel and check your settings again.

9. Save the configuration for WebSphere then restart the server.

Testing the connection

When the server starts, launch the Administrative Console, it should ask for the user name and password for authentication. This is because Global Security is enabled. Provide the user name and password as `cn=wasadmin,o=itso` and password as `password`. If you are able to login successfully that means your configuration is working fine.

> **Note:** if the uid field is set for the user in the LDAP directory, then you can use the uid for user name for authentication, in this case: `wasadmin`.

For the IBM SecureWay Directory configuration and sample user and group registration refer to Section 10.13, "Connecting to directory servers (LDAP)" on page 315.

10.4.3 Custom Registry

WebSphere can be configured to use a type of user registry other than LocalOS and LDAP. This registry is referred to as a Custom Registry. It will be necessary to provide a Java class that provides WebSphere with a standard interface in order for WebSphere to communicate with the registry in an appropriate fashion. Refer to Chapter 8.3, "CustomRegistry SPI" on page 183 for information regarding the development of such an interface.

Open the Custom Registry window in the Admin console by selecting **Security -> User Registries -> Custom**.

| Active User Registry | Custom ▼ | ⓘ Specifies the active User Registry when security is enabled. |

Figure 10-9   Custom Registry is set for Active User Registry

1. Enter a user name and password. This is the identity under which the request is sent to the registry

2. Click **OK**.

Figure 10-10   Custom Registry user name and password

3. If the custom registry requires properties to be set, then click the **Custom Properties** link

4. Click **New.**

5. The name of the property and its value must be added although additional information may also be added as required. The information entered here will be passed to the custom registry implementation during initialisation.

6. Click **OK.**

7. If there are no errors at this stage, select **Security -> Global Security**. Ensure that the Active User Registry option is set to Custom and that Global Security is enabled. If this is not the case, make the necessary changes and then click **OK.**

8. The changes will need to be saved and WebSphere restarted.

10.5 SWAM

The SWAM (Simple WebSphere Authentication Mechanism) is intended for simple, non-distributed, single application server type runtime environments. The single application server restriction is due to the fact that SWAM does not support forwardable credentials. For more information on authentication mechanisms, refer to Section , "Authentication mechanisms" on page 224.

Using SWAM does not require further configuration for WebSphere Application Server V5, you can simply select SWAM as the authentication mechanism on the global security page, as shown on Figure 10-11.
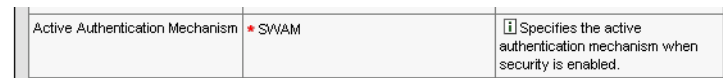
Figure 10-11   Configuring SWAM for the application server

Once you enable security, WebSphere Application Server will use the currently set authentication mechanism.

10.6 LTPA

Lightweight Third Party Authentication (LTPA) is intended for distributed, multiple application server and machine environments. It supports forwardable credentials therefore it supports Single Sign-On. LTPA is able to support security in a distributed environment through the use of cryptography.

LTPA requires that the configured User Registry is a central shared repository such as LDAP or a Windows Domain type registry.

For more information on authentication mechanisms, refer to Section , "Authentication mechanisms" on page 224.

10.6.1 Single Sign-On

Single Sign-On is the process wherein users provide their credentials, user identity, password and/or token, once within a session. These credentials are available to all enterprise applications for which Single Sign-On was enabled without prompting the user to reenter user name and password.

The goal is for an enterprise to be able to have one network identity per user allowing the centralized management of the various roles the user may have in different applications, so correct rules can be applied without duplication of either user data or requiring multiple identities for the one user.

In practise, network identity management is not yet mature enough within most enterprises to achieve a single user registry, particularly when legacy applications are exposed to the Web. Different application servers have typically implemented their own security or utilized the operating environment security of the platform on which they have been deployed. The task is then to authenticate a user and provide, within the current session, a credential which can be passed through to, and understood by each application.

IBM has previously developed The Lightweight Third Party Authentication, (LTPA) mechanism enabling Single Sign-On between various application servers. A token, the transient cookie *LtpaToken*, is generated by the authenticating server, for this book we configured WebSphere, WebSeal and Domino in different scenarios to provide the LTPA token for Single Sign-On. The cookie is encrypted using LTPA keys which must be shared amongst all Single Sign-On participating servers, and contains user authentication information, the network domain in which it is valid for Single Sign-On and an expiry time.

The token is issued to the Web user in a cookie called a transient cookie; this means that the cookie resides in the browser memory, is not stored on the user's computer system and expires when the user closes the browser. This cookie is easily recognized by its name: *LtpaToken*.

The public domain has largely adopted Kerberos technology to provide the same functionality.

Tivoli Access Manager, with its reverse proxy security sever, WebSeal, provides a more robust mechanism for Single Sign-On which can be used in conjunction with LTPA and TAI (Trust Association Interceptor) provided as the Trust Association Mechanism (TAM). Access Manager can integrate most back-end applications servers using the Global Sign-On mechanism to third party user registries or extensions to the TAM schema to include legacy user identities and passwords.

In a scenario presented in this chapter we will use LTPA for enabling single sign-on. For how to use Tivoli Access Manager WebSeal together with LTPA please refer to Chapter 12, "Tivoli Access Manager" on page 367. The requirements for enabling Single Sign-On using LTPA are:

► All Single Sign-On participating server have to use the same user registry (for example LDAP server).

► All Single Sign-On participating servers must be in the same DNS domain (cookies are issued with a domain name and will not work in another domain than for which it was issued).

► All URL requests must use domain names. No IP addresses or hostnames are allowed as this will cause the cookie not to work properly.

► Browser must be configured to accept cookies.

► Servers time and time zone must be correct. Single Sign-On token expiration time is absolute.

All servers participating in the Single Sign-On scenario must be configured to share LTPA keys.

10.6.2 Configuring LTPA for WebSphere

The following steps will show you how to configure LTPA for WebSphere Application Server.
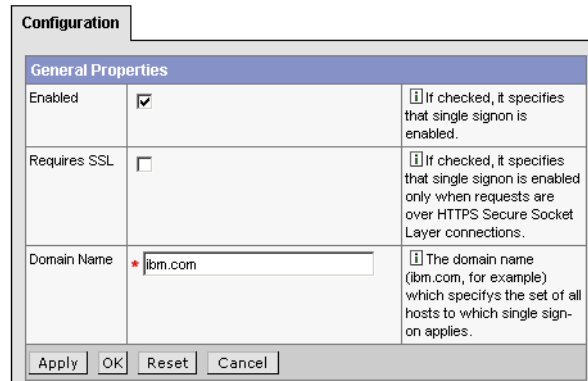
1. Open LTPA configuration panel. Launch WebSphere Administrative Console and expand the tree **Security -> Authentication Mechanisms -> LTPA.**

2. Specify the following attributes:

   Password is the password to protect LTPA keys. You will need this password in order to import the keys into any other SSO enabled server. Confirm the password by retyping it in the Confirm Password field.

   Timeout specifies the amount of time in minutes for which LTPA token will be valid without re-authentication. For the purpose of the test you can leave this field default. We have entered the value 30.

   Click **OK** to accept configuration, Key file name you will specify after setting up Single Sign-On attributes.

3. Save the configuration for WebSphere to make the changes effective.

4. Configure the Single Sign-On panel by clicking the link **Single sign-on (SSO)** at the bottom of the LTPA page.

Figure 10-12   SSO Configuration panel for LTPA

5. Check the **Enabled** box, if it is not already checked.

6. Leave the **Require SSL** check box unchecked for now.

7. Specify the Domain Name, in our example we have set this field to ibm.com.
    This domain is used when HTTP cookie is created for Single Sign-On and
    determines the scope to which Single Sign-On applies.

> **Important:** Remember that all SSO enabled servers must be in the same
> DNS domain. All URLs must use domain name and not IP addresses.

8. Click **OK** to approve changes.

9. Save the configuration for WebSphere.

10.6.3 Generate LTPA keys

The following steps will show you how to generate the LTPA keys for the
WebSphere Application Server.

1. On the LTPA Configuration panel click the button **Generate Keys**. This will
    launch key generation process in the background. You will be prompted to
    save the configuration after the process is completed.

2. Save the configuration for WebSphere to have the generated keys stored in
    the WebSphere configuration, they will appear in the security.xml file.

3. Re-open **LTPA** configuration page.

4. Specify the **Key File Name** which is the name of the file where LTPA keys will
    be stored when you export them. You need to export the keys in order to

enable Single Sign-On on another server. Specify the full path name for the key file. We have used `c:\WebSphere\Appserver\etc\SSO_ltpakeys.`

5. Click **Export Keys**, Keys that have been exported in our scenario are presented in the example below.

Example 10-1   Contents of the key file generated from WebSphere LTPA panel

```
#IBM WebSphere Application Server key file
#Tue Aug 13 18:25:07 EDT 2002
com.ibm.websphere.CreationDate=Tue Aug 13 18\:25\:07 EDT 2002
com.ibm.websphere.ltpa.version=1.0
com.ibm.websphere.ltpa.3DESKey=FDspFou4xxe1m4Il84JmAk+EXLb1QclZp7ji+BJPSDM\=
com.ibm.websphere.CreationHost=wassrv01
com.ibm.websphere.ltpa.PrivateKey=9qo7ytSCbTf/62bvAyExobRikGAwF4vE/vKnKe7K8OeJa
/jUoiAtyeo6rQumiUw/otwCBSaGWWvAHAwpTKR3CP7oJm4CAxyjOUVNF2B2iSZspH+ekZ+fS62Amp64
HT+ppljshfmyjX4WZAOxRQdKpvHvX3BUMU1BjuRnlpQqp2Pov/VlBqpnSJI5vcLRrXZDCNUEA4KdOCH
cKyq5H22Iox4PiZ4rvpZ5UCXdjxfcAOrUbw+5KK1eZdVQLrcxHb/ufBQ51RrA6m2R8PCZua26RUOJwi
x1YOJpGBuwKNeKDCq/pY4l7OK4nkyOEXrq7EBlOVkhtC7JEsR4o5Mbc1JSbuyCJsRamjgX5/plEFZSB
HE\=
com.ibm.websphere.ltpa.Realm=dirsrv01.itso.ibm.com\:389
com.ibm.websphere.ltpa.PublicKey=AO/uOSd3vL4zo7VUN3k8VSw9F+zpgwbRnDHmi8G8gmm5Tb
CKGonK4Hl+gQ9dzSDNgkDJ3BWYJEkrCj77oZsI4RCZZk1RexDqLByEO9ffR/WyT7PR4FaMMFaZoOIha
DX3GyF3yHov6l3/DcsrvYCLgO3Fc+SPsX/QnHPDQOXyKZ6lAQAB
```

As you can see in the example, three types of keys have been generated for LTPA.

► The private key, used for the LTPA server to sign the LTPA Token.

► The public key, used to verify the digital signature.

► A shared key, used to encrypt/decrypt those tokens.

10.6.4 Enable LTPA Authentication for WebSphere

The following steps will show you how to enable LTPA for WebSphere Application Server.

1. Select **Security -> Global Security** in the Administrative Console.

2. Make sure that **Active Authentication Mechanism** is set to **LTPA (Light weight Third Party Authentication).**

During the previous steps we assumed that the user registry is already configured for LDAP. Configuring the User Registry is covered in previous sections. LTPA also works with other user registries, only this example is using LDAP, however for Single Sign-On scenarios you might want to use centralized user registries.

> **Note:** The generation of the LTPA keys must be performed when the LDAP
> server settings are configured; this guarantees that the LDAP host name is
> present in the exported file, as shown in bold in the previous example.
> Domino needs this information during the Web SSO Configuration
> Document creation process.

3. Save the configuration for WebSphere.

Your WebSphere Application Server is now configured to use LTPA
authentication mechanism. You will need to log out from the Administrative
Console and restart the server in order for the changes to take effect.

10.7 JAAS Configuration

JAAS for WebSphere Application Server can be configured using the
Administrative Console. JAAS provides the pluggable authentication mechanism
for WebSphere. If you want to know more about JAAS refer to Section 8.6,
"JAAS" on page 204.

10.7.1 Application Login Information

WebSphere allows you to configure the pluggable authentication module for your
application server.

1. Launch the Administrative console then login with administrative privileges.

2. Select **Security -> JAAS Configuration -> Application Logins**, to get to the
   page show on Figure 10-13.
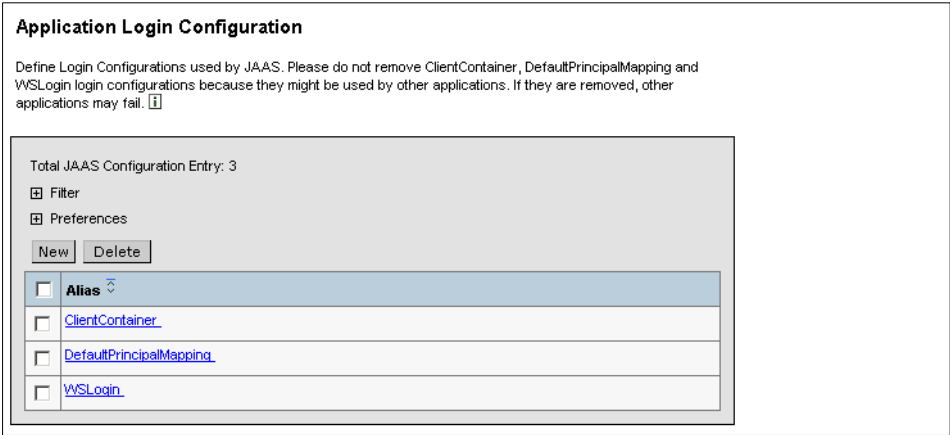
Figure 10-13   Application Login Configuration

3. Each Login module defines a module class which is the implementation of the JAAS login module itself. Select a login module, **WSLogin** for example, then click the **JAAS Login Modules** link, to get the following picture Figure 10-14.
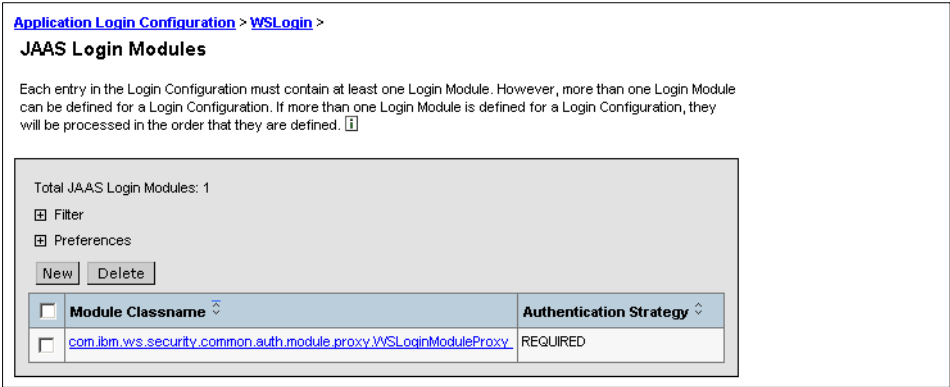
Figure 10-14   JAAS login module configuration

When you create your own login module you will have to create a new entry and configure it as it was shown in the previous steps.

The JAAS configuration for the server includes another element, which is the *wsjaas.conf* file under the <WebSphere_root>\properties direcotry. It defines the JAAS login modules for the JVM according to the JAAS specification; for example:

```
WSLogin {
```

```
      com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProxy
required delegate=com.ibm.ws.security.common.auth.module.WSLoginModuleImpl;
};
```

The example above tells the JVM which is the login module for the WSLogin alias. The Java code in the application will refer to this alias to invoke the login module defined for JAAS.

There is another configuration file provided for the Java clients, the *wsjaas_client.conf* file under the <WebSphere_root>\properties directory.

10.7.2 J2C Authentication Data Entries

J2C Authentication entries provide an easy way of administering user name and password pairs for authentication purposes for any resources in WebSphere Application Server V5. These entries associated with alias names; where the alias names can be used in the resource definitions to refer to a certain user name and password pair.

Following steps will explain how to set up a J2C Authentication Data Entry using the Administrative Console.

1. Click **Security -> JAAS Configuration -> J2C Authentication Data Entries**.

2. When you click **New**, the page will appear, specify the user ID and password that may be used by Java 2 Connector or WebSphere Application Server V5.0 DataSource.

3. Each user ID and password set is identified by a unique alias name. Enter `itsobankds_auth` as the alias, `dbuser` is the user ID and `password` is the password.

Figure 10-15   J2C Authentication entry in the Administrative Console

4. Save the configuration.

The J2C Authentication Entries are stored in the security.xml file under the <WebSphere_root>\config\cells\<your_node> directory. The password fields are encoded in the file.

For more information on J2C security refer to Section 7.3, "J2C Security" on page 169.

10.8 Configuring SSL

The SSL implementation used by the application server is the IBM Java Secure Sockets Extension (JSSE). The JSSE is a set of Java packages that enable secure Internet communications. It implements a Java version of the SSL and TLS protocols and includes functionality for data encryption, server authentication, message integrity and client authentication. Configuring JSSE is very similar to configuring most other SSL implementations (for example, GSKit); however, a few differences are worth noting.

► JSSE allows both signer and personal certificates to be stored in a SSL key file, but it also allows a separate file to be specified called a trust file. A trust file can contain only signer certificates. Therefore, all personal certificates can be stored in an SSL key file and all signer certificates stored in a trust file

► JSSE does not recognize the proprietary SSL key file format that is used by the plug-in (.kdb files); instead, it recognizes standard file formats such as

JKS (Java Key Store). As such, SSL key files cannot be shared between the plug-in and application server and a different implementation of the key management utility (ikeyman) must be used in order to manage application server key and trust files

10.8.1 SSL configurations

The first step in configuring SSL is to define an SSL configuration repertoire. A repertoire contains the details necessary for building an SSL connection, such as the location of the key files, their type and the available ciphers. WebSphere provides a default repertoire called *DefaultSSLSettings*.

From the Admin console, select **Security -> SSL** to see the list of SSL repertoires.

The appropriate repertoire is referenced during the configuration of a service that sends and receives requests encrypted using SSL, such as the Web and EJB containers.

Follow the steps below to configure a new entry in the SSL repertoire.

1. From the SSL Configuration Repertoires page, click **New.**
2. Enter an alias by which this configuration will be know.
3. Cick **OK.**
4. Select the new SSL entry by clicking the link and then click the **Secure Sockets Layer (SSL)** link in Additional Properties.

   The new configuration details can be entered in the window that appears.

SSL Configuration Repertoires > DefaultSSLSettings >

New

Specifies the list of defined Secure Socket Layer configurations. i

Configuration

General Properties

Alias	★ sampleSSL	i Specifies one of the Secure Socket Layer configurations in the repertoire to use.
Key File Name	${USER_INSTALL_ROOT}/etc/sample	i The fully qualified path to the key file that contains public keys and perhaps private keys.
Key File Password	********	i The password for accessing the key file.
Key File Format	JKS	i The format of the key file.
Trust File Name	${USER_INSTALL_ROOT}/etc/sample	i The fully qualified path to a trust file containing the public keys.
Trust File Password	********	i A password for accessing the trust file.
Trust File Format	JKS	i The format of the trust file.
Client Authentication	☐	i Client authentication is supported by the CSIv2 authentication protocol only.
Security Level	HIGH	i Selects from a preconfigured set of security levels.
Cipher Suites	SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA SSL_DHE_DSS_WITH_DES_CBC_SHA SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA Add >> << Remove	i When specified, overrides the setting of Security Level.
Cryptographic Token	☐	i Enables/disables crypto hardware support.

Apply   OK   Reset   Cancel

Figure 10-16   An SSL configuration window

5. Enter the location of the key file name. For details regarding key files, refer to Chapter 10.9, "Demo keyfile" on page 260.

6. Enter the password for the key file.

7. Select the appropriate key type.

8. Enter the location of the trust file name. For details regarding trust files, refer to Chapter 10.9, "Demo keyfile" on page 260.

9. Enter the password for the trust file.

10. Select the appropriate trust type.

11. If client authentication is supported by this configuration, then check the Client Authentication box. This will only affect HTTP and LDAP request. In

order to support client authentication for IIOP requests refer to Chapter 10.12, "SSL between the Java client and WebSphere" on page 308.

12. The appropriate security level must be set. Valid values are low, medium and high. Low specifies only digital signing ciphers (no encryption), medium specifies only 40-bit ciphers (including digital signing), high specifies only 128-bit ciphers (including digital signing).

13. If the preset security level does not define the required cipher, it can be manually added to the cipher suite option.

14. Check the Cryptographic Token box is hardware or software cryptographic support is available. Refer to the InfoCenter for details regarding cryptographic support.

15. Additional properties can be added by selecting the Custom Properties link in the Additional Properties section.

16. Click **OK** to apply the changes.

17. If there are no errors, save the changes to the master configuration and restart WebSphere.

More details can be found on using the SSL definitions in Chapter 10.11, "SSL between the Web server and WebSphere" on page 300 for HTTP requests and Chapter 10.12, "SSL between the Java client and WebSphere" on page 308 for IIOP requests.

10.9 Demo keyfile

SSL relies on the existence of digital certificates. A digital certificate reveals information about its owner, such as their identity. During the initialisation of an SSL connection, the server must present its certificate to the client in order for the client to determine the servers identity. The client may also present the server with its own certificate in order for the server to determine the clients identity. SSL is, therefore, a means for propagating identity between components.

The Application Server provides a set of certificates that may be used for testing purposes. However, the identities contained in the certificates are generic and the expiration dates are set artifically low. This section describes the process for creating digital certificates that are tailored for use in a production system.

A client can trust the contents of a certificate if that certificate has been digitally signed by a trusted third party. Certificate Authorities (CA) act as a trusted third party and will signed certificates on the basis of their knowledge of the certificate requestor.

WebSphere supports the concept of two types of key store which are refered to as a *key file* and a *trust file*. A key file contains a collection of certificates, each one of which may be presented during an SSL connection initiation in order to prove identity. Incidently, a key file will also contain the associated private key for each certificate. A server will manage at least one key file, although a client may also manage one. A trust file contains a collection of certificates that are considered trustworthy and with which the presented certificate will be matched against during an SSL connection initiation in order to assure identity. A client will typically manage at least one trust file, although a server may also manage one. See Figure 10-17 on page 261.

Splitting the certificates into two files: key file and trust file; increases security. The certificate stores are essential parts of the secure communication since the certificates provide the base for trust. The most sensitive element is the private certificate, the one that is presented for identification. This certificate must be secured carefully, and once it is stored in the keystore protected by a password it should not be opened again. While on the other side there is the list of signer certificates subject to change, new signers or trust parties may need to be added, which means that the store needs to be opened.
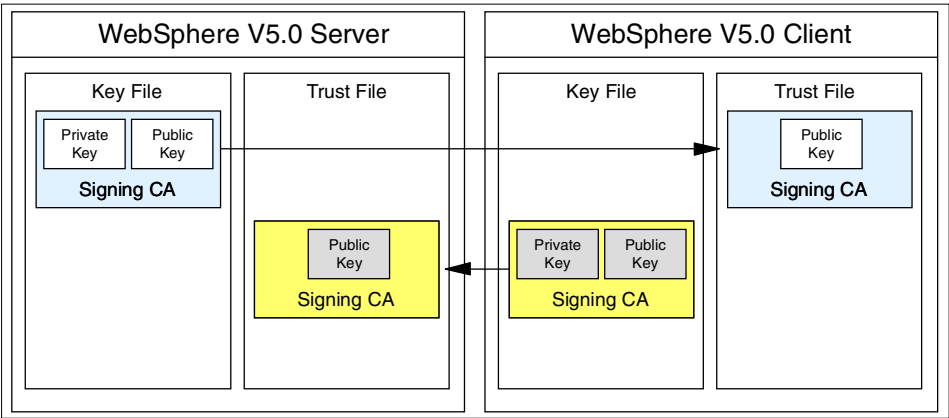
Figure 10-17   Correlation between server and client key stores

Whilst this demonstrates how the two types of key store may be used, it is also possible to combine the key and trust files. WebSphere provides the following key stores in the <WAS_ROOT>/etc directory.

Table 10-3   WebSphere default key stores

File	Description
DummyServerKeyFile.jks	server-based key file
DummyServerTrustFile.jks	server-based trust file

File	Description
DummyClientKeyFile.jks	client-based key file
DummyServerTrustFile.jks	client-based trust file

The key store type in this case is Java Key Store (JKS), a format that is supported by both WebSphere and the supplied key generation utility, ikeyman. This utility will be used in the section to generate a new certificate. There are, generally, two options when deciding how to create a new certificate.

► Request that a CA generates the certificate on your behalf. This will probably involve providing enough information so that the CA can validate the identity of the certificate requestor. The CA will create a new certificate, digitally sign it and then deliver it to the requestor in, presumably, a secure fashion. Popular Web browsers are pre-configured to trust certificates that are signed by certain CA's and so no further client configuration is necessary in order for a client to connect to the server (that this certificate relates to) via an SSL connection. Therefore, CA-signed certificates are useful where configuration for each and every client that will access the server is impractical

► Generate a self-signed certificate. This may well be the quickest option and will probably require fewer details in order to create the certificate. However, the certificate will not be signed by a CA. This may prove troublesome in certain cases. Every client that is likely to receive this certificate, in other words any client that will connect to this server over an SSL connection, will need to be configured to trust the signer of this certificate. Since the certificate has been self-signed, the signature is not likely to be in the client's trust file and so must be added. If access to every client is impractical then this configuration will simply not occur. Therefore, self-signed certificates are only useful when each of the clients can be configured to trust the certificate

Note: It is technically possible in some cases to present a self-signed certificate to an untrusting client. In some Web browsers, for instance, when the certificate is received and is found to not match any of those listed in the client's trust file, a prompt will appear asking if the certificate should be trusted for the connection (or even added to the trust file).

The file used by a Java client to refer to a key store is the SAS properties file. Refer to Chapter , "The sas.client.properties file" on page 103 for details on the client SAS file. The WebSphere server, on the other hand, stores the key store information in the repository and the key stores are referred to in the security.xml file. Therefore, all server-side configuration should be done through the administration tools, such as the Administrative Console.

10.9.1 Generating a self-signed certificate

The process for creating a self-signed certificate is relatively straight-forward. An understanding of public/private key pairs and of PKI will be useful. The result of this process is a key store that will replace the temporary key store provided by WebSphere, on both server and client side.

The server's key file

The following steps will describe how to generate a new self-signed certificate with the IBM ikeyman utility.

1. Launch the ikeyman tool. It may be started from the command line in the bin directory as `ikeyman.bat` (on Windows platforms) or `ikeyman.sh` (on UNIX platforms)
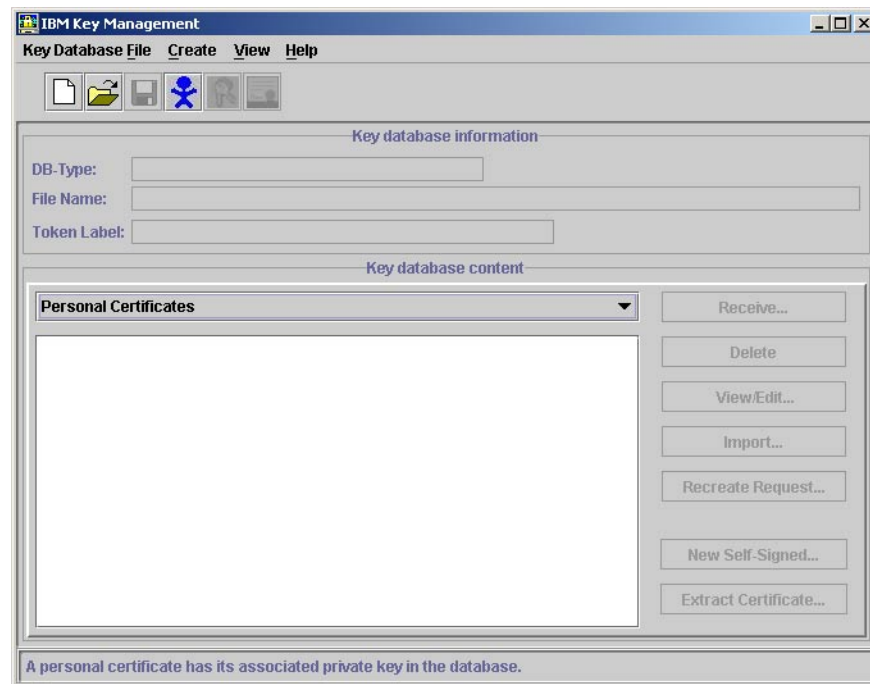

Figure 10-18   ikeyman key management utility

2. From the menu bar, select **Key Database File -> New**

3. Ensure that the Key database type is set to **JKS**. This section will use the following file names to represent the appropriate key stores

   WASV5ServerKeyFile.jks - server key file

   WASV5ServerTrustFile.jks - server trust file

WASV5ClientKeyFile.jks - client key file

WASV5ClientTrustFile.jks - client trust file

4. Enter **WASV5ServerKeyFile.jks** as the file name

5. Enter the directory that will hold the key file as the location, in this case: c:\was\etc.
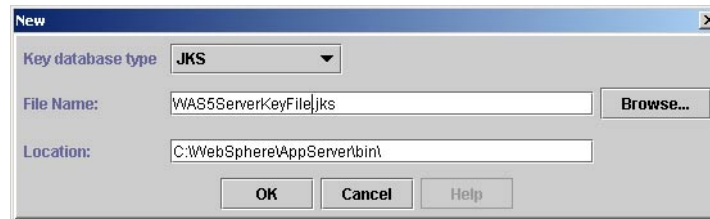
6. Click **OK.**

Figure 10-19   Saving the new key store

7. A password prompt will appear. Enter a password and repeat to confirm. This password will be required to read from or write to this file in the future, so do not forget it. The password strength is determined by the variety of the characters used in the password.
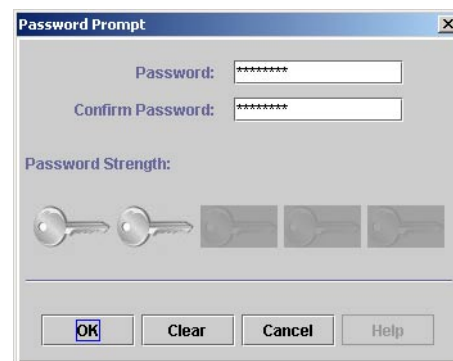
8. Click **OK**.

Figure 10-20   Preventing unauthorised access to key store with a password

9. A list of signer certificates should appear which represents the identities of a selection of CA's. This list is not needed and so can be deleted by selecting each certificate, clicking **Delete** and confirming.
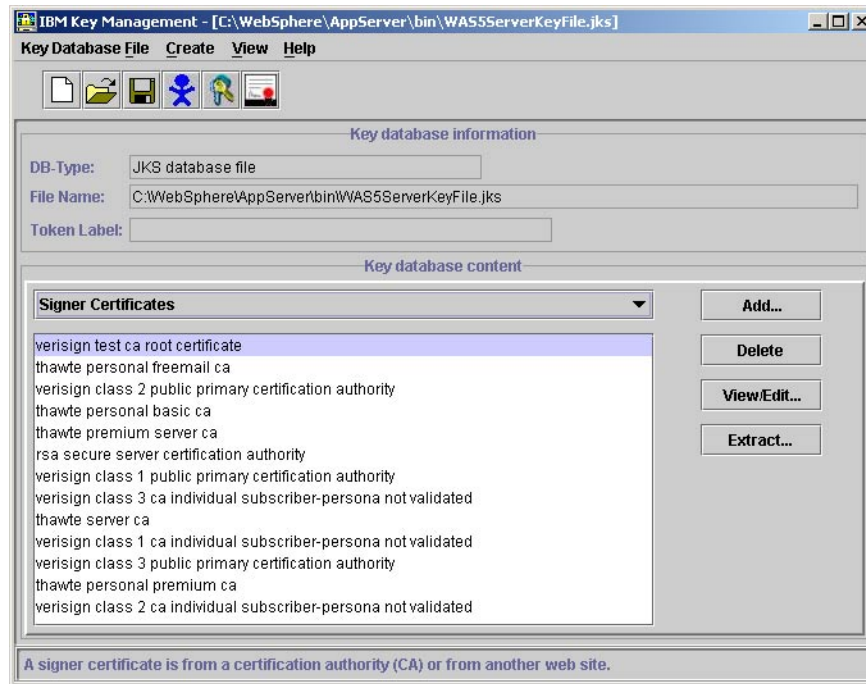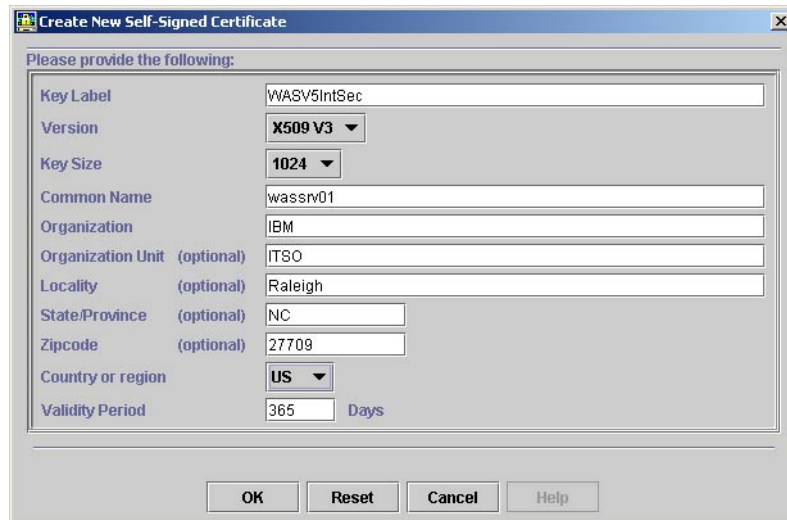
Figure 10-21   The default list of signer certificates

10. Now a self-signed certificate can be generated. From the menu bar, select
Create -> New Self-Signed Certificate.

11. A window will appear requesting information in order to generate the
certificate. Enter **WASV5IntSec** as the Key Label. Ideally, spaces should not
be used in the key label.

12. Select **X509 V3** as the Version and **1024** as the Key Size.

13. The first two fields of the Distinguished Name, those being Common Name
and Organization, are mandatory. The WebSphere server name might be
entered as the Common Name, for example: **wassrv01**.

14. Whilst Organization Unit, Locality and State/Province are optional fields, it is
recommended that appropriate values are entered; for example: **ITSO**,
Raleigh, **NC**.

15. Select the appropriate Country, in our case: **US**.

16. Enter **365** as the Validity Period.

17. Click **OK**.

Figure 10-22   Provide details for the new self-signed certificate

18. The new self-signed certificate should be added to the Personal Certificates
    list.
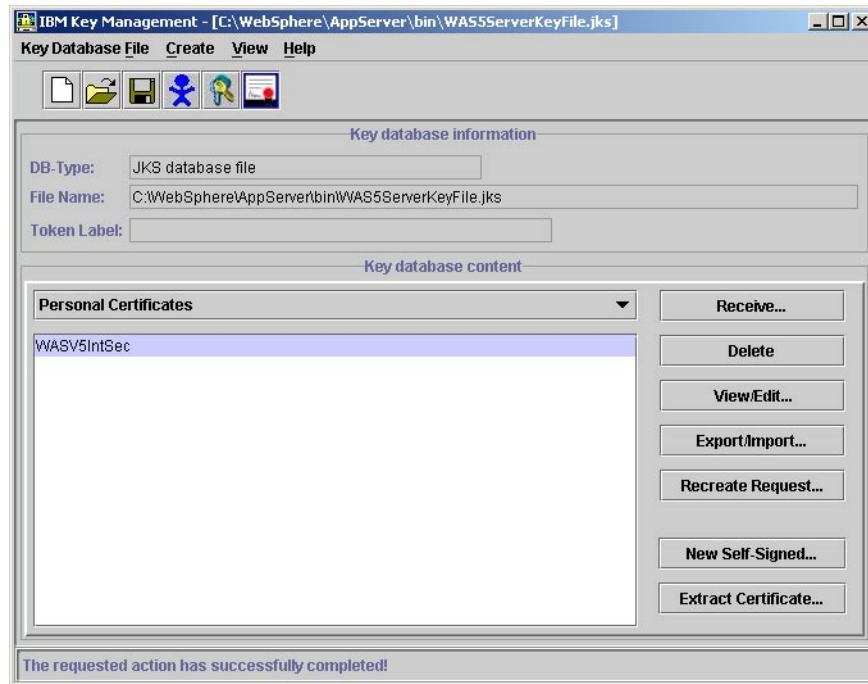
Figure 10-23   The new self-signed certificate

19. It is necessary to extract this certificate so it can be later added to the client's trust file. Click **Extract Certificate.**

20. Select **Base64-encoded ASCII data** as the Data type.

21. Enter **WASV5IntSecPubCert.arm** as the Certificate file name.

22. Enter the directory that will hold the extracted certificate as the Location, in our case: **<WebSphere_root>\etc**.

23. Click **OK.**

Figure 10-24   Extract certificate facility

24. From the menu bar, select **Key Database File -> Close**. This will close the current key store.

The server's trust file

To create the server's trust file, repeat the first nine steps from Chapter , "The server's key file" on page 263 (up to the clicking **OK** on the password prompt) with a file name of WASV5ServerTrustFile.jks. It is not necessary to populate the trust file with any certificates at this stage.

The client's key file

The client's key file provides a client certificate during the SSL connection initialisation. This certificate contains the identity of the caller that is not necessarily restricted to establishing an SSL connection but may also be used for authentication purposes at a J2EE level. The creation of this key file is very similar to that of the server's key file and so refer to Chapter , "The server's key file" on page 263 for details. The file name of the key file is WASV5ClientKeyFile.jks in this sample, the certificate label is WASV5ClientSec and the extracted certificate is WASV5ClientSecPubCert.arm. The client certificate can be added to the server's trust file.

1. Open **WASV5ServerTrustFile.jks** in ikeyman.
2. Select **Signer Certificates** from the Key Database Content drop-down menu
3. Click **Add.**
4. Enter the details for the client certificate (WASV5ClientSecPubCert.arm)
5. Click **OK.**
6. Enter the label for the certificate which is **WASV5ClientSec.**
7. Click **OK**. The certificate should be added to the list of signer certificates.
8. Close the file.

The client's trust file

To create the client's trust file, repeat the first nine steps from Chapter , "The server's key file" on page 263 (up to the clicking **OK** on the password prompt) with a file name of WASV5ClientTrustFile.jks. It will be necessary to add the server's extracted certificate as a signer certificate. The process for this is documented in "The client's key file" previous section.

There should now be the four key stores called:

▸ WASV5ServerKeyFile.jks
▸ WASV5ServerTrustFile.jks
▸ WASV5ClientKeyFile.jks

► WASV5ClientTrustFile.jks

There should also be two extracted certificates called

► WASV5IntSecPubCert.arm

► WASV5ClientSecPubCert.arm

10.9.2 Requesting a certificate signed by a CA

The ikeyman tool can be used to generate a certificate request. A certificate will be required for the server and one for each client. The process documented below is for the server's key file although the process will be similar for every certificate with only minor changes needed between each.

1. Launch the ikeyman tool. It may be started from the command line in the bin directory as `ikeyman.bat` (on Windows platforms) or `ikeyman.sh` (on UNIX platforms)

2. From the menu bar, select **Key Database File -> New**.

3. Ensure that the Key database type is set to JKS. This section will use the following file names to represent the appropriate key stores

   – WASV5ServerKeyFile.jks - server key file

   – WASV5ServerTrustFile.jks - server trust file

   – WASV5ClientKeyFile.jks - client key file

   – WASV5ClientTrustFile.jks - client trust file

4. Enter **WASV5ServerKeyFile.jks** as the file name.

5. Enter the directory that will hold the key file as the location, in this case: `c:\WebSphere\Appserver\etc`.

6. Click **OK**.

7. A password prompt will appear. Enter a password and repeat to confirm. This password will be required to read from or write to this file in the future, so do not forget it. The password strength is determined by the variety of the characters used in the password

8. Click **OK.**

9. From the menu bar, select **Create -> New Certificate Request.**

10. Enter a Key Label which will identify this key. It is recommended that spaces are not used. The value used in this section is **WASV5IntSecPubCert.**

11. Select the appropriate key size. **1024** is an appropriate default value.

12. Enter a common name, which may be the name of the WebSphere server that this certificate will be owned by. It is important that the common name contains the fully-qualified name for the server.

13. Enter an Organization, for example: **IBM**.

14. Organization Unit, Locality and State/Province are optional but it is recommended that values are provided, for example: **ITSO**, **Raleigh**, **NC**.

15. Select the appropriate country, in this case: **US**.

16. The file name in which the certificate request is to be stored in should be entered in the last field. The value used in this section is **<WebSphere_root>\etc\servcertreq.arm** (for a Windows machine).

17. Click **OK.** A message confirming the creation of the certificate request should be returned.
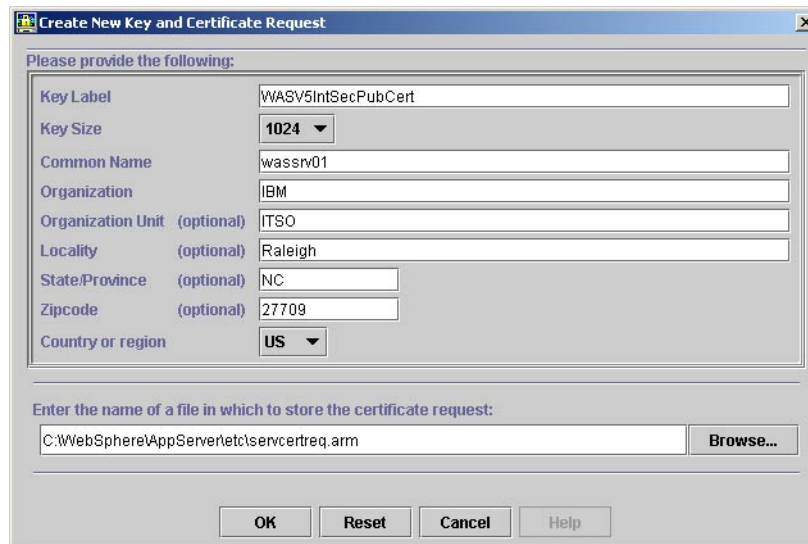

Figure 10-25   A certificate request information window

18. Click **OK**.

Figure 10-26   Certificate request confirmation message

> **Note:** Consult the documentation from your chosen CA prior to completing the certificate request fields. For example, VeriSign Server IDs stipulate that the Command Name (CN) must represent the fully qualified server name.

Viewing the contents of the certificate request file (servcertreq.arm in this sample) shows that a certificate has been generated.

Example 10-2   A certificate request generated by ikeyman

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBmzCCAQQCAQAwXDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAk5DMRAwDgYDVQQHEwdSYWxlaWddo
MQwwCgYDVQQKEwNJQkOxDTALBgNVBAsTBElUUO8xETAPBgNVBAMTCG1rYTJ2YWxxvMIGeMAOGCSqG
SIb3DQEBAQUAA4GMADCBiAKBgHHsF7RWcLXGF6DPY3KnFJTHnONmf/Ni2lhURbJkgnSTl2x2vECe
rrQ5qhYI7mXX4v1zL4FSDM9TzMCz8V4P5FXAwjyJRlPODfSxMP9h/kIJWiAx2n1X2FnHiKcVAz17
EE27hVObMTfj47Ww4ydQ7JMQFy1C7pZnHuJL3Ga1qBZLAgMBAAGgADANBgkqhkiG9wOBAQQFAAOB
gQAmzZ+9bsrqLjCw8tsxkUJHG7SMCspv1UCmc447aRubkM717j6nnwOzr9QgCO8bxzvOD6C35Liv
MDPrc5ML+CT9KVHtXnV9mpOKx9+3d1A4YDAdEoQAOcPYXu9n6aDfG69ZIdwjBMlohsy7q8qPlnGd
yqfmhhEbFcn+PlW86bhnjg==
-----END NEW CERTIFICATE REQUEST-----
```

It is clear that the process of creating a certificate request actually creates the certificate. That means the public/private keypair exist in the JKS file. The certificate request is actually requesting that this certificate is signed by the CA. This certificate request can be view in ikeyman by selecting Personal Certificate Requests from the Key Database Content drop-down box.
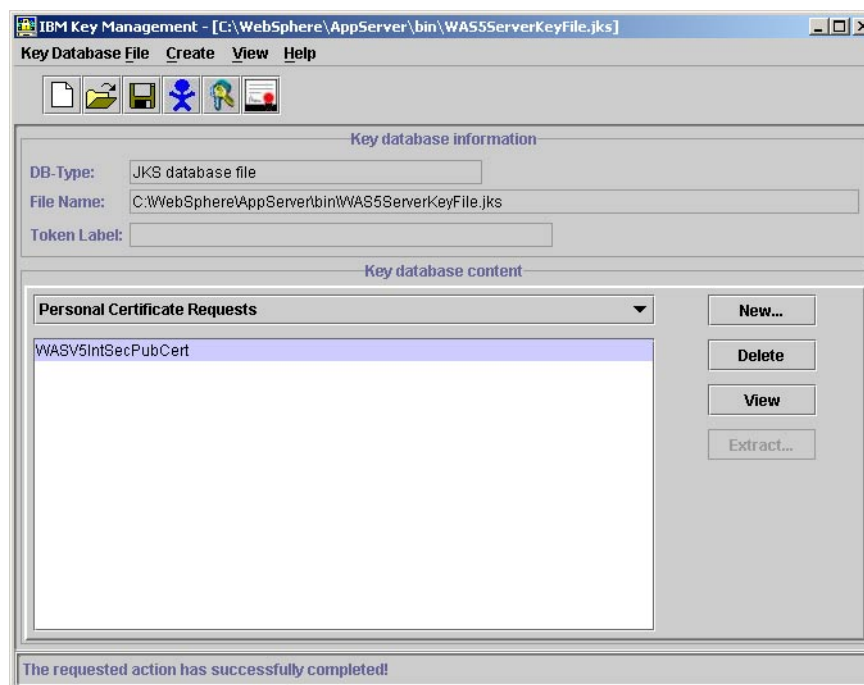
Figure 10-27   A personal certificate request

Note: Don't destroy this certificate request. It must be present in the key store in order for the certificate to be added once it has been signed by the CA.

Of course, the CA will need to validate the identity claimed by the certificate before signing it. The certificate request should be copied into the form provided by the CA, usually on their Web site and submitted.

Note: Be sure to submit the request in its original form. If the request is being pasted from a text editor, then ensure that extra characters, such as spaces, have not been added to the ends of the lines. If the request is not submitted correclty, the reply may not import correctly.

Some CA's offer a convenient service for generating *test* certificates. These are certificates that are valid for a short period of time, such as a month and are intended for testing purposes only. If the certificate request used in this section is submitted to Thawte, a well-known CA, the reply is in Base64 encoded format.

Example 10-3   Certificate reply from Thawte

```
-----BEGIN CERTIFICATE-----
MIICgDCCAemgAwIBAgIDLqKsMAOGCSqGSIb3DQEBBAUAMIGHMQswCQYDVQQGEwJa
...
..
.
XmYOnq8HX/fjOil6NQxW48bp3O8=
-----END CERTIFICATE-----
```

This reply should be save into a plain text file, then import into the key store that created the original certificate request.

> **Note:** As with the request, ensure that the CA's reply is copied to a file correctly. Ensure there are no additonal characters appended to the end of lines that would otherwise affect the import process.

1. Ensure that ikeyman has the relevant key file open and select **Personal Certificates** from the Key Database Content dropdown list.

2. Click **Receive**.

3. Enter the file name of the reply from the CA.

4. The reply will probably be encoded with Base64, so ensure Base64-encoded ASCII data is selected from the Data Type drop-down list.

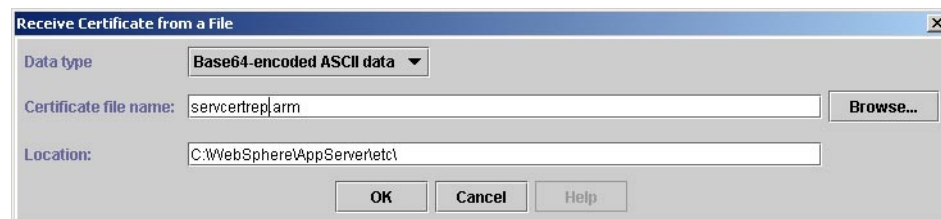5. Click **OK**.

Figure 10-28   Importing a CA reply

Assuming that the appropriate certificate request can be found, the personal certificate will be added to the personal certificate list. Its contents can be viewed by clicking **View/Edit.**

10.9.3 Using the Java keytool

Another way to create self-signed keys, create certificate requests is to use the Java keytool command line utility that comes with the Java Development Kit from Sun. The Java keytool utility gives your more flexibility to create your own customized certificate request with the DN (Distinguished Name) of your choice.

For more information about Java keytool, refer to the documentation at
http://java.sun.com/j2se/1.3/docs/tooldocs/tools.html#security

10.9.4 Configuring WebSphere to use a key store

Once a key store has been configured, either by creating a self-signed certificate or by creating a certificate request and importing the reply, WebSphere can be configured to make use of the certificates. WebSphere will use the certificate in order to establish a secure connection with a client via SSL.

> **Note:** Before making changes to the sas.client.props file, it is recommended to make a copy for restoration purposes if the need arises.

It is necessary to define an SSL Configuration, which will be used to determine how SSL connections are established with the appropriate WebSphere components.

Using the Administrative Console

The following steps will create a new SSL definition entry for WebSphere using the Administrative Console, follow the steps from Section 10.8.1, "SSL configurations" on page 258 using the values below:

1. Enter **WASV5IntSec** as the Alias.

2. Select the new **WASV5IntSec** link.

3. Select **Secure Socket Layer (SSL).**

4. Enter the location of the server's key file in the Key File Name text area**: c:\WebSphere\Appserver\etc\WASV5ServerKeyFile.jks** in this example .

5. Enter the key file password in the Key File Password text area

6. Ensure that **JKS** is the selected Key File Format

7. Enter the location of the server's trust file in the Trust File Name text area, in our case: **c:\WebSphere\Appserver\etc\WASV5ServerTrustFile.jks**.

8. Enter the trust file password in the Trust File Password text area

9. Ensure that **JKS** is the selected Trust File Format

10.Ensure the Client Authentication is **enabled**. This is optional and is enabled in this example.

11.Ensure the Security Level is set to **High.**

12.Ensure that the Cryptographic Token box is not ticked.

13.It is not necessary to provide any custom properties. Click **OK**.

14.Save the changes to the master configuration by selecting the link at the top of the window.

The password is stored in a file called security.xml in <WebSphere_root>/config/cells/<your_cell> and is protected with an Base64 ASCII encoding.

Example 10-4   Excerpt from security.xml

```
<repertoire xmi:id="SSLConfig_3" alias="WASV5IntSec">
   <setting xmi:id="SecureSocketLayer_1"
   keyFileName="C:\WebSphere\AppServer\etc\WASV5ServerKeyFile.jks"
   keyFilePassword="{xor}KzosK25tbA==" keyFileFormat="JKS"
   trustFileName="C:\WebSphere\AppServer\etc\WASV5ServerTrustFile.jks"
   trustFilePassword="{xor}KzosK25tbA==" trustFileFormat="JKS"
   clientAuthentication="true" securityLevel="HIGH"
   enableCryptoHardwareSupport="false"/>
</repertoire>
```

Note: Although the password appears as a series of asterisks in the Admin console, it will be stored in an easily readable string in the repository. It is a simple task for an eavesdropper to decode this string and recover the password and so it is important to protect the respository from unauthorised users

The appropriate WebSphere components may now be set to use the newly-defined SSL configuration. It might also be necesary to configure some non-WebSphere components, such as a Web server in order to ensure a secure connection between all components. Typically, a digital certificate will be created for each component. In a Web server scenario, for instance, the WebSphere server will own a certificate and the Web server will own another. The certificates will identify the particular component it is owned by.
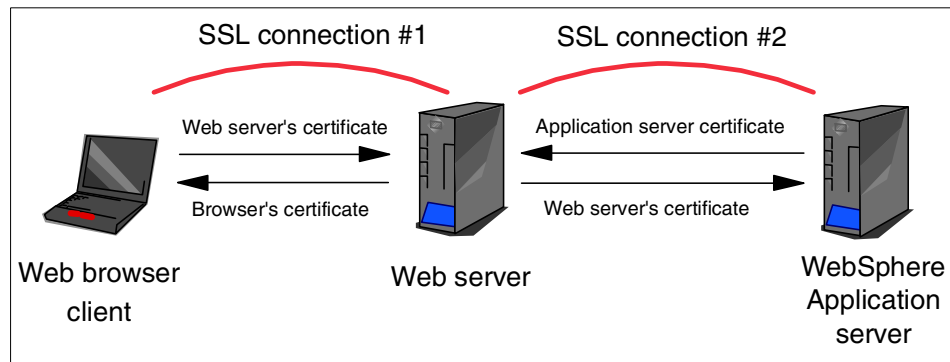
Figure 10-29   How certificates are distributed during an SSL connection initialisation

Figure 10-29 shows how certificates are exchanged during an SSL *handshake*. An SSL handshake occurs whilst the connection between the two components is being established and it is during this time that the identities of the server and the client, should this option be selected, are transferred. A single SSL connection will not span multiple servers and so the first SSL connection in this example exists between the Web browser and the Web server. The second SSL connection, which initiates a second handshake, exists between the Web server and the WebSphere Application Server. During the first handshake, the Web server's and Web browser's certificates are exchanged and during the second handshake, the Application Server's and Web server's certificates are exchanged.

Using wsadmin

It is possible to add an SSL configuration to the repository using the wsadmin tool, rather than the Administrative Console. Refer to Chapter D, "Using wsadmin scripting for security configuration" on page 509 for further information on this topic and some sample scripts.

10.10 SSL between the Web client and the Web server

IBM's HTTP Server (IHS), as of version 1.3.24, supports SSL version 3 and version 2 and TLS version 1. Whilst IHS is based on the Apache Web server, it is necessary to use the IBM-supplied SSL modules, rather than the OpenSSL varieties. This section will describe configuration of the IHS, although it is entirely possible that another supported Web server is used in its place.

SSL is disabled by default and it is necessary to modify a configuration file and generate a server-side certificate using the ikeyman tool provided with IHS in order to enable SSL.

10.10.1 Generating a digital certificate

The use of ikeyman to create a digital certificate has been documented in Chapter 10.9, "Demo keyfile" on page 260. However, the version of ikeyman provided with the IBM HTTP Server supports the CMS Key Database format (KDB) which is not supported with the IBM JSSE-based ikeyman tool provided with WebSphere. The KDB format is required by the IBM HTTP Server and therefore the ikeyman usage differs slightly to begin with. The differences will be documented in this section.

Be sure to start the correct version of ikeyman, do not start the JSSE version provided by WebSphere. The process for starting ikeyman differs depending on the type of operating system used.

For UNIX systems, the process is as follows.

1. issue the following command: `export JAVA_HOME=/usr/jdk_base`

2. From the <IHS_ROOT>/bin directory, run `./ikeyman`

For Windows systems, the process is as follows.

1. From the Start menu, select **Programs -> IBM HTTP Server 1.3.24 -> Start Key Management Utility**

> **Note:** This is a different version of the ikeyman utility. It works with different type of key stores.
>
> You can see the difference between the two applications, the IBM HTTP Server ikeyman does not have the small icon with the human shape.
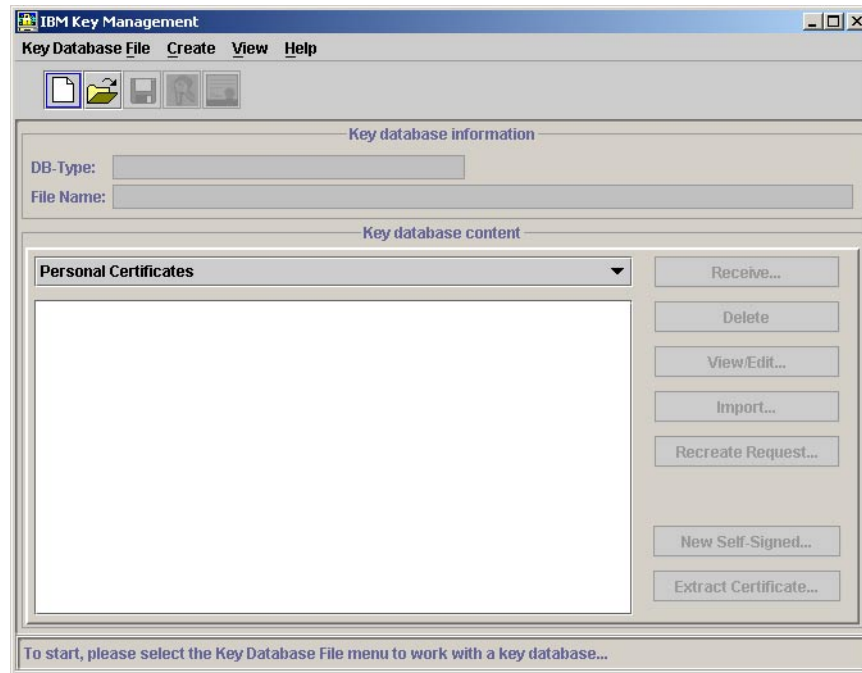
Figure 10-30   The IBM HTTP Server ikeyman utility

2. From the menu bar, select **Key Database File -> New**

3. Select **CMS key database file** from the Key database type drop-down menu

4. Enter a file name for the new key store, **IHS1324Certs.kdb** in this example

5. Enter a path in the Location text area, **c:\IBMHttpServer\conf\keys** in this example.

6. Click **OK.**

7. Enter a password in the Password text area and again in the Confirm Password area.

8. **Enable** the Stash the password to a file option, which will allow IBM HTTP Server to make use of the password to gain access to the certificates contained in the key store.

9. Click **OK.**

10. Click **OK** to confirm that the password has been stashed.

Once the key store file has been created, the process is the same as documented in Chapter 10.9, "Demo keyfile" on page 260, which explains how to use the ikeyman utility to create self-signed certificates (see Chapter 10.9.1, "Generating a self-signed certificate" on page 263) and certificate requests that should be submitted to a CA (see Chapter 10.9.2, "Requesting a certificate signed by a CA" on page 269).

11. Once the required certificates have ben generated and exported, close the ikeyman utility.

In addition to the key database files and the previously generated Certificate Signing Request (CSR) file, if this option was taken, the .sth suffixed file, which contains the password stash and the .crl and .rdb files, which contain internal information specific to the CSR should all be copied as a precautionary measure.

10.10.2 Configuring the IBM HTTP Server

The following section will show you how to enable security for IBM HTTP Server using the Administration console. You can also edit the httpd.conf file manually to enable SSL and do other modifications.

The steps described below only work for IBM HTTP Server, other Web servers have different administration interfaces and different implementations of administering security; however the process and the to-dos should be the same for every Web server.

The httpd.conf file

This file, located in the <ihs_root>/conf directory, provides configuration information for the Web server, such as the location of specific files and modules to be loaded. The IBM SSL module will need to be refered to in this file. This file will have been largely configured during the installation of WebSphere and so few changes need to be made.

> **Note:** Another file called http.conf.sample is provided in the same directory which contains many more configuration options than the original httpd.conf file, including a reference to the IBM SSL module. It is recommended, assuming that no changes have already been made to the original httpd.conf file, to copy the http.conf.sample file to httpd.conf and then make changes as required.

The IBM HTTP Server Administration console

A alternative mechanism for configuring the IHS is to the supplied Administration utility. This section describes how to configure SSL using this utility, which is Web-based. Before accessing the tool, it may be necessary to set an administration user name and password.

1. From the command line, change to the <ihs_root> directory

2. Check for the existence of the admin.passwd file in the conf directory and if it does not exist, run

   ```
   htpasswd -c conf/admin.passwd <admin_user>
   ```

   where <admin_user> is the user name of the administrative user, for example: `ihsadmin`.

3. Enter the administrator's password and repeat to confirm.

4. Start the IBM HTTP Server. On windows issue the `apache` command on UNIX systems run the `apachectl` script from the IBM HTTP Server bin directory.

5. Open a Web browser and enter the following URL

   ```
   http://<server_name>
   ```

   where <server_name> is the host name of the server running the IHS process.

6. The IBM HTTP Server splash screen comes up; select the **Configure Server** link.

7. Enter the IBM HTTP Server administrator's user name and password in the dialog box; Click **OK**.
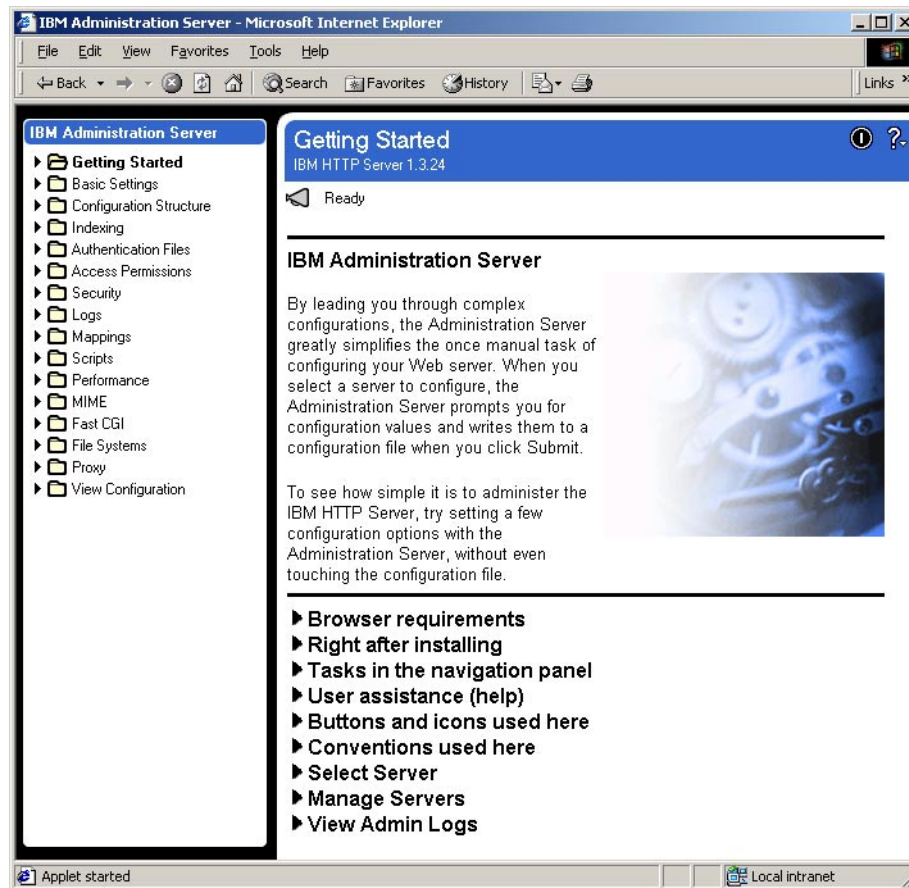
Figure 10-31   Administrative Console for IBM HTTP Server

8. Once the Web interface has loaded, select **Basic Settings -> Core Settings** in the left hand frame.

9. Ensure the Server Name contains a valid fully-qualified host name. If it does not, enter the relevant value and click Submit which can be found at the bottom of the window.

10. Select **Basic Settings -> Advanced Properties.**

11. In the Specify additional ports and IP addresses section, click **Add.**

12. Enter **443** in the Port text area. This is the default server port for SSL connections.

13. Click **Apply.**

14. Click **Submit** at the bottom of the window.

15. Select **Basic Settings -> Module Sequence**. This presents a list of modules loaded by the IBM HTTP Server and should include the IBM WebSphere AppServer module, which is probably at the top of the list. It is necessary to add the IBM SSL module.

16. Click **Add.**

17. From the Select a module to add drop-down list, select **ibm_ssl(IBMModuleSSL128.dll)**.

18. Ensure that the Module dynamic link library is set to modules/IBMModuleSSL128.dll.

19. Click **Apply.** This will add the module to the list.

20. Click **Submit.**

21. Every operation so far has been performed at the Global level. Now a new VirtualHost will be created specifically for SSL connections. Select **Configuration Structure -> Create Scope**.

22. Select **VirtualHost** from the first drop-down list.

23. Enter a fully-qualified name or IP address in the Virtual host name text area; in our example: **websrv01.itso.ibm.com**.

24. Enter **443** in the Virtual host port text area.

25. Enter the server name in the Server name text area, in our case: **websrv01.itso.ibm.com**.

26. For the server path, enter the path to the **<ihs_root>/htdocs** directory

27. Click **Submit.** A new VirtualHost entry should appear in the right-hand frame.

28. Select **Security -> Server Security.**

29. Ensure the scope is set to **Global**. If not, click **Scope** and select **Global.**

30. Enable SSL by clicking **Yes.**

31. Enter the path and name of the key store created with the ikeyman utility.

32. Enter an SSL version 2 session ID timeout of **100**.

33. Enter an SSL version 3 session ID timeout of **1000**.

34. Click **Submit.**

35. Select **Security -> Host Authorization**.

36. Ensure the Scope is set to the VirtualHost defined previously: websrv01.

37. Enable SSL by clicking **Yes.**

38. Click **Submit.**

39. Restart the IBM HTTP Server. This can be performed from the command line or by clicking the **Restart Server** icon in the top-right corner of the

Administrative Server's main window. Ensure that there are no error messages, which will need resolving before starting the next stage.

This concludes the basic IBM HTTP Server configuration for SSL. The result can be tested by loading a Web browser and entering the URL: `https://<server_name>` ; where <server_name> is the host name of the server.

A message may appear from the browser explaining that the Web browser is attempting to establish a secure link with the server, like on the next picture Figure 10-32 in Microsoft Internet Explorer.
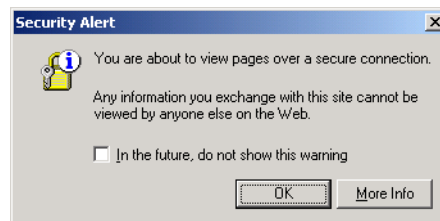

Figure 10-32   Preparing for a secure connection in Internet Explorer

During the initilisation of the secure connection, the IBM HTTP Server will provide a certificate to the Web browser. If this certificate has not been signed by any of the CA's known to the Web browser, then a message may appear providing details of the certificate and requesting if the certficate should be trusted.
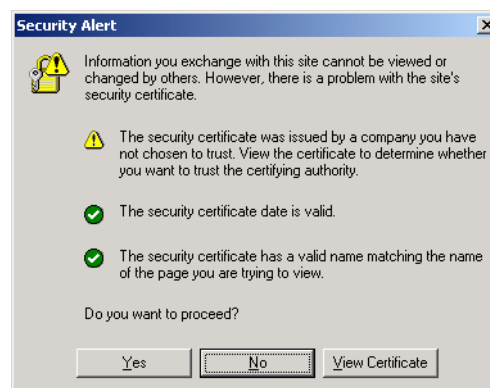

Figure 10-33   Unauthenticated certificate

To see the details of the certificate provided by the server, click the **View Certificate** button. In the details you will find information about the certificate itself, who issued the certificate and to whom; also you can follow the certificate path of the issuers.

In the case of a self-signed certificate, like in our example, the issued to and the issued by fields will be the same, and the certificate path only has one level indicating that the root certificate is untrusted. Realistically these certificates should not be trusted; however, for testing purposes it should be acceptable to trust the certificate.

Proceed by accepting the certificate and the Web page should load.

IBM HTTP Server Cipher Support

To ensure the Web server operates at the highest possible level of security, SSL connections should be restricted to specific ciphers. The mod_ibm_ssl library supports the following cipher specifications. For more details on the cipher support refer to the configuration file, httpd.conf.sample in the <IHS_ROOT>\conf directory. You will find the following cipher suites listed there:

SSL Version 2 Cipher Specifications in the format of: short name (HEX code), long name, description

```
27, SSL_DES_192_EDE3_CBC_WITH_MD5,          Triple-DES (168-bit)
21, SSL_RC4_128_WITH_MD5,                   RC4 (128-bit)
23, SSL_RC2_CBC_128_CBC_WITH_MD5,           RC2 (128-bit)
26, SSL_DES_64_CBC_WITH_MD5,                DES (56-bit)
22, SSL_RC4_128_EXPORT40_WITH_MD5,          RC4 (40-bit)
24, SSL_RC2_CBC_128_CBC_EXPORT40_WITH_MD5,  RC2 (40-bit)
```

SSL Version 3 Cipher Specifications in the format of short name (HEX code), long name, description

```
3A, SSL_RSA_WITH_3DES_EDE_CBC_SHA,          Triple-DES SHA (168-bit)
33, SSL_RSA_EXPORT_WITH_RC4_40_MD5,         RC4 SHA (40-bit)
34, SSL_RSA_WITH_RC4_128_MD5,               RC4 MD5 (128-bit)
39, SSL_RSA_WITH_DES_CBC_SHA,               DES SHA (56-bit)
35, SSL_RSA_WITH_RC4_128_SHA,               RC4 SHA (128-bit)
36, (See  SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5,   RC2 MD5 (40-bit)
32, SSL_RSA_WITH_NULL_SHA
31, SSL_RSA_WITH_NULL_MD5
30, SSL_NULL_WITH_NULL_NULL
```

TLS Version 1 Cipher Specifications in the format of short name (HEX code), long name, description

```
62, TLS_RSA_EXPORT1024_WITH_RC4_56_SHA,     RC4 SHA(56 Bit)
64, TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA,    DES SHA(56 bit)
```

Configuring the IBM HTTP Server to use specific ciphers is a matter of adding the required ciphers to the cipher specification list. From the IHS Administration utility, following these steps.

1. Select **Security -> Host Authorization**

2. Click **Scope** and select the VirtualHost configured earlier: websrv01.

3. Click the **Add** button associated with the Cipher Specification list.

4. Select the required cipher and click **Apply.**

5. Repeat until all required ciphers have been added to the list. More than one cipher specification may be added in order to make the chances of a secure connection with a client more likely.

6. Click **Submit** at the bottom of the window.

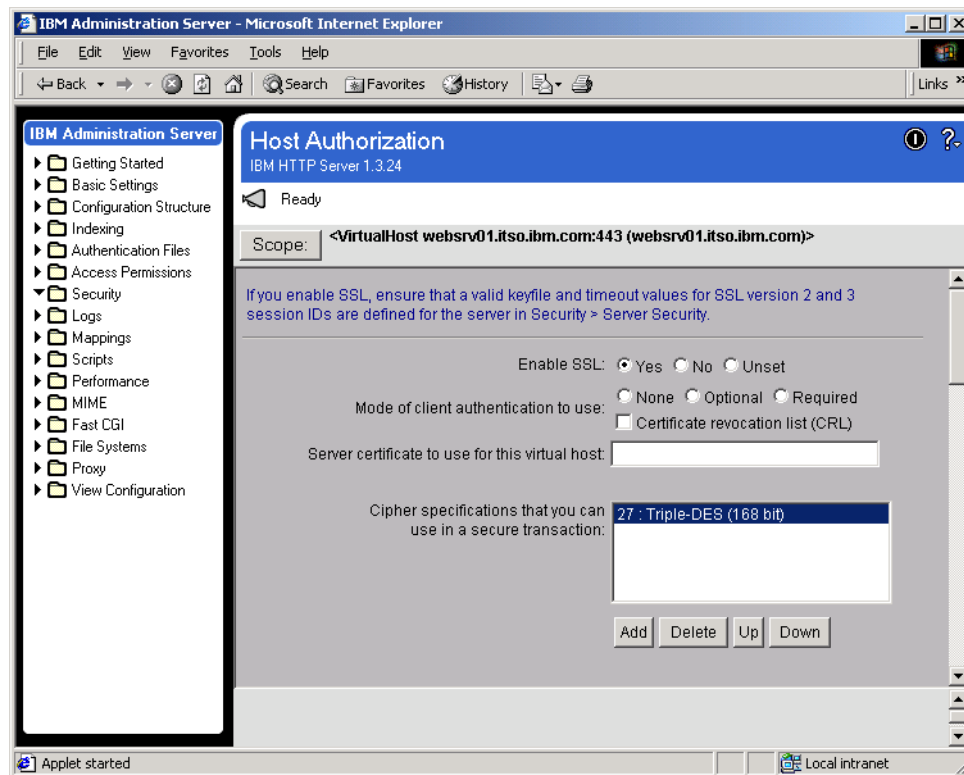7. Restart the Web server to ensure the changes will take effect.

Figure 10-34   Selecting ciphers

Note: If the Cipher Specification list is empty, then the IHS will have all specifications available to use as required.

A Web browser may be used to test the cipher specifications. If the Web browser does not support one of the ciphers in the Cipher Specification list, an error may be displayed and the Web page will not load.

Some Web browsers provide the ability to view the type of cipher used to secure the connection.

If the Web browser is Internet Explorer, once the Web page has loaded, from the menu bar, select **File -> Properties**. The connection details are mentioned on the appearing panel.
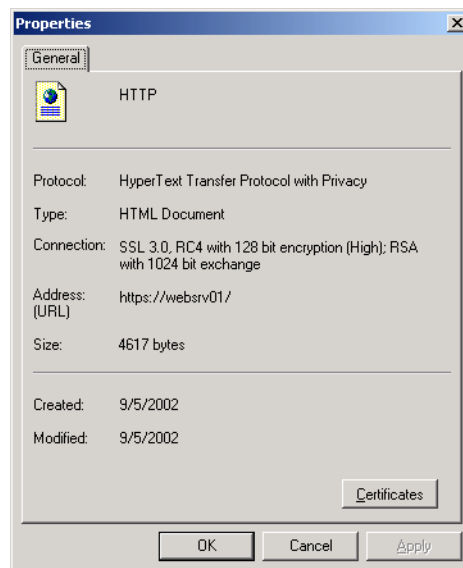

Figure 10-35   Certificate properties

Note: If the Web browser has SSL and TLS disabled, it will not be able to establish a secure link with the IHS. In the case of Internet Explorer, SSL can be enabled.

► From the menu bar, select **Tools -> Internet Options**

► Click the **Advanced** tab and scroll down to the Security section

There should be an option to enable SSL 2.0, SSL 3.0 and TLS 1.0. By default, SSL versions 2 and 3 are enabled.

10.10.3 Client-side certificate for client authentication

This section discusses, how to use client side certificates with your Web server and with your WebSphere Application Server. It will also show how to configure your servers to support client-side certificates and use them as a base of user authentication.

Obtaining a personal certificate

The Web client may also provide a digital certificate in order to assert an identity during an SSL initialisation. Typically, the creation of a client-side certificate involves a CA. Alternatively, the IBM Tivoli SecureWay PKI package or a similar product from another vendor may be used to implement a PKI solution. This involves the overhead of managing the PKI infrastructure, as well as creating the individual certificates for each authenticating user.

The process for requesting and installing a personal client-side certificate on Windows is documented in this section.

For demonstration purposes, the free Personal Certificate Program offered by Thawte Consulting was used. The process for requesting a personal certificate will differ from CA to CA, with each providing different facilities.

From the Thawte Web site, `http://www.thawte.com`, select the option to receive a free personal email certificate and fill out the necessary forms. Be sure to request an X.509v3 certificate and that the email address entered is valid and can be used. The process is relatively straight-forward and a certificate will be issued within a matter a minutes of registration; we got the certificate issued, a notification was sent by Thawte about the fact, we went to the Thawte Web site to pick up the certificate. At the end of the process we installed the certificate into the Web browser, which was Microsoft Internet Explorer in this case.

Make sure that the certificate is installed and you have the right certificate. The following steps will show how to check the certificate in Microsoft Internet Explorer.

1. From the Internet Explorer menu bar, select **Tools -> Internet Options -> Content -> Certificates.**

2. Select **Client Authentication** from the Intended purpose drop-down list.

3. Click the **Personal** tab. The certificate should be displayed in the list.
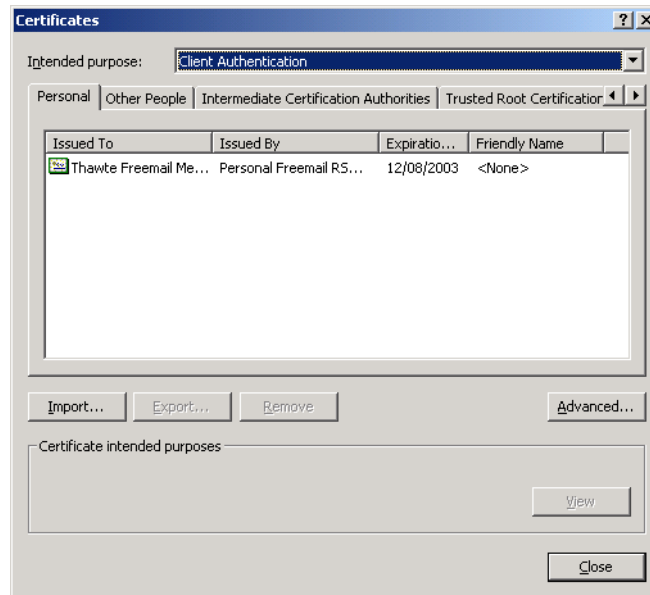
Figure 10-36   A personal certificate

4. Select the certificate and click **View**. This displays the certificate's properties, including the certificate path. The path shows that *Thawte Personal Freemail CA* is the top-level CA and since this is a trusted root CA the certificate can be easily accepted during an SSL connection initialisation.
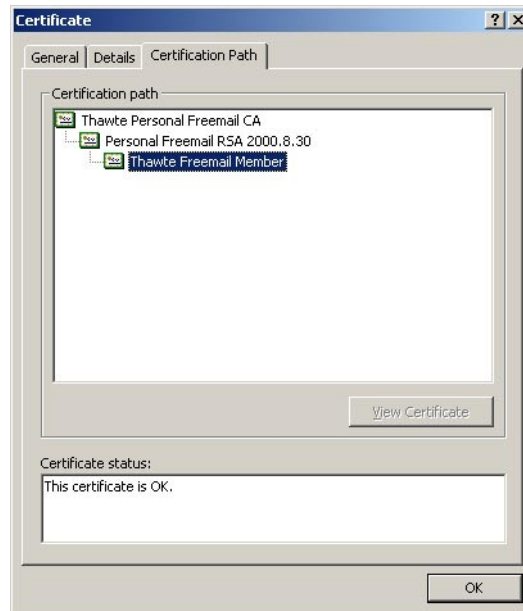
Figure 10-37   The certificate chain

The following checks are also recommended on any certificate installed into Microsoft Internet Explorer, for use as a client side certificate. Double-click any certificate entry and verify the following details.

► Under the General tab, the certificate's intended use includes: *Proving your identity to a remote computer (required)*

► Under the General tab: *You have a private key that corresponds to this certificate (required)*

► Under the Certificate Path, the Certificate status is given. If you are presented with the message:

> *This CA Root certificate is not trusted because it is not in the Trusted Root Certification Authorities Store*

then you must install the corresponding signing root CA certificate in the Certification Authorities Store.

Figure 10-38 show the resulting personal certificate generated when using the Thawte Freemail Certificate Program. The certificate subject Distinguished Name (DN) includes two components; the e-mail entity (E) and the Common Name (CN) Thawte Freemail Member.
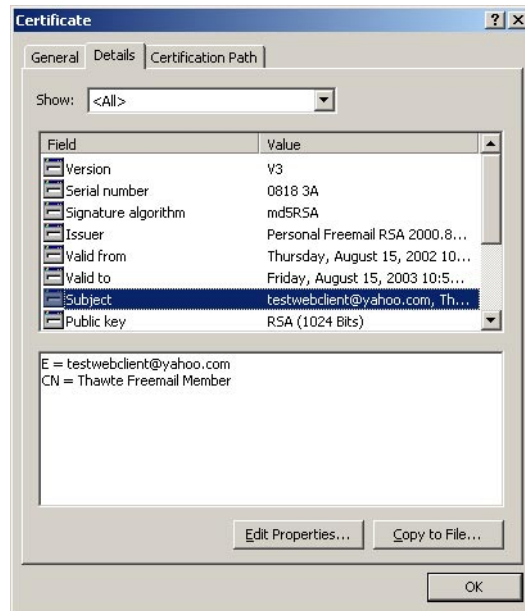
Figure 10-38   Certificate details

This certificate may be enabled in order to provide a client certificate when accessing Web pages over an SSL connection, with the `https://` URL prefix.

LDAP advanced security settings

Certificate based authentication requires either that WebSphere map the entire certificate subject Distinguished Name (DN) to a like LDAP Distinguished Name or that WebSphere certificate filtering be used to map a certificate subject Distinguished Name to a specific LDAP field for a given LDAP user.

> **Note:** As structure and hierarchy are of concern when managing an LDAP directory, it is not always possible to use the same Distinguished Name (DN) that is supported by the client side certificates.

Using the WebSphere LDAP Certificate Filter option

This section assumes that you have successfully installed a personal certificate into a client Web browser and that you have previously enabled WebSphere Global Security authenticating users against a remote LDAP Directory Server. It is anticipated that the personal certificate subject Distinguished Name (DN) does not necessarily match, in any way, your LDAP Distinguished Name (DN).

In the following sample we will use Thawte's personal certificate requested through the Free Certificate Program.

Take a look at the certificate details in Figure 10-38 on page 290, the Subject attribute of the certificate equates the certificate SubjectDN, and the value in our case is:

```
E = testwebclient@yahoo.com
CN = Thawte Freemail Member
```

If you used an alternative PKI solution, the subjectDN will be different, but equally unique, with the issuer (signer) value being different.

Another alternative to see the SubjectDN for a certificate is using the Java keytool utility. Export the public certificate form the browser using the Base-64 encoded format for the export, then run the following command:

```
keytool -printcert -file <exported_certificate_file>
```

The result for our example was:

```
Owner: EmailAddress=testwebclient@yahoo.com, CN=Thawte Freemail Member
Issuer: CN=Personal Freemail RSA 2000.8.30, OU=Certificate Services,
O=Thawte, L=Cape Town, ST=Western Cape, C=ZA
Serial number: 8183a
Valid from: Thu Aug 15 10:56:15 EDT 2002 until: Fri Aug 15 10:56:15 EDT
2003
Certificate fingerprints:
    MD5:  C5:55:B4:CD:42:19:3D:A2:54:F0:66:E7:20:31:CE:3D
    SHA1: D0:14:77:5F:8E:0B:FB:80:57:CD:F7:7E:49:DF:7C:52:FE:20:2B:67
```

The SubjectDN is the value of the Owner attribute, which is:

```
EmailAddress=testwebclient@yahoo.com, CN=Thawte Freemail Member
```

The next step is to modify WebSphere LDAP filtering rules to map the certificate subjectDN field to the IBM SecureWay LDAP uniqueIdentifier field for a given user. You do not necessarily have to use the SecureWay LDAP uniqueIdentifier field. However, you should ensure that the data type of the field selected is capable of handling the specific value and 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.

Configure WebSphere to use certificate mapping
The following steps will show you how to configure WebSphere Application Server to use the certificate filter as required.

1. Login to the WebSphere Administration Console.

2. Select **Security -> User Registries -> LDAP**.

3. Select the **Advanced LDAP Settings** at the bottom of the LDAP page.

4. Set the following fields on the Configuration panel:

   Certificate Map Mode: **CERTIFICATE_FILTER**

   Certificate Filter: **uniqueIdentifier=${SubjectDN}**

5. Click **OK**, then save the configuration for WebSphere.

6. You have to stop and start the application server to implement the advanced LDAP modifications.

Configure the directory server to use certifcate mapping

The directory server storin your user registry has to be updated to reflect the new values to use certificate mapping. Basically the uniqueIdentifier field has to contain the SubjectDN for each user, the SubjectDN value can be extracted from the public certificate of the user.

In the following steps we will use the IBM SecureWay LDAP Directory.

1. Launch the SecureWay Directory Management Tool.

2. Rebind as an Authenticated User with adequate privileges to modify user credentials.

3. Expand the directory tree and select the user entity against which you wish to authenticate the personal client certificate. In this example let us use the user: manager.

4. click **Edit**, switch to the **Other** tab and find the uniqueIdentifier field.

5. Enter the SubjectDN value for the uniqueIdentifier from the certificate. Use the value returned by the Java keytool utility, in this case:

   ```
   EmailAddress=testwebclient@yahoo.com, CN=Thawte Freemail Member
   ```

Figure 10-39   Setting the uniqueIdentifier for the user

6. Click **OK** when you are done.

7. Repeat this step for each individual user against which you wish to perform client side certificate authentication, setting the appropriate certificate string in the uniqueIdentifier field each time.

Modifying the Web server to support client certificates
You must ensure that the selected Web server is configured to request client side certificates. In the following example we will use the IBM HTTP Server to show how to configure SSL for the Web server to force the clients to send their certificates.

1. The Web server requires SSL enabled andconfigured in order to use client side certificates. Follow the steps in Section 10.10.2, "Configuring the IBM HTTP Server" on page 279 to enable SSL for your IBM HTTP Server if you have not done it yet.

2. There are two possibilities to set client-side certificates required: using the Administrative console for IBM HTTP Server; or editing the httpd.conf file manually, which is the fastest way, considering that we only need to add one more directive.

3. If you decide the use the IBM HTTP Server Administrative Console. Login to the IBM HTTP Server Administrative Console as it is documented in Section 10.10.2, "Configuring the IBM HTTP Server" on page 279.

4. Select **Security -> Host Authorization** from the left side navigator.

5. click the **Scope** button, and select you virtual host from the list that appears in a new window, in our case: **<VirtualHost websrv01.itso.ibm.com:443>**. The new setting should appear next to the Scope button.

6. Change the Mode of client authentication to use to **Required**.

7. **Submit** the changes with the button at the bottom.

8. Restart the Web server.

> **Note:** If you choose to edit the httpd.conf file manually; open it with your favorite browser from the <IHS_ROOT>\conf directory then find the SSL configuration part. It should start with the definition of a new VirtualHost, for example: `<VirtualHost wassrv01.itso.ibm.com:443>`. Find the SSLEnable directive then insert the following directive:
>
> `SSLClientAuth required`
>
> Save the httpd.conf file, then close it and finally restart the Web server.

9. WebSphere Application Server does not support the port 443 by default, you have to modify the default host configuration. Login to the WebSphere Administration Console, then select: **Environment -> Virtual Hosts**, then click **default host**.

10. Select **Host aliases**, click **New**, then provide the following values:

    Host Name: *

    Port: `443`

    Click **OK** when you are finished.

11. Save the configuration for WebSphere.

12. You have to stop and start the server to make the changes effective.

Testing the client side certificate

The best way to test the client certificate is to use the Default Application that ships with WebSphere and use the snoop servlet by accessing it with you Web browser. Access the following address from the client: https://<your_server_name>/snoop, to determine if your browser is correctly passing a client certificate.
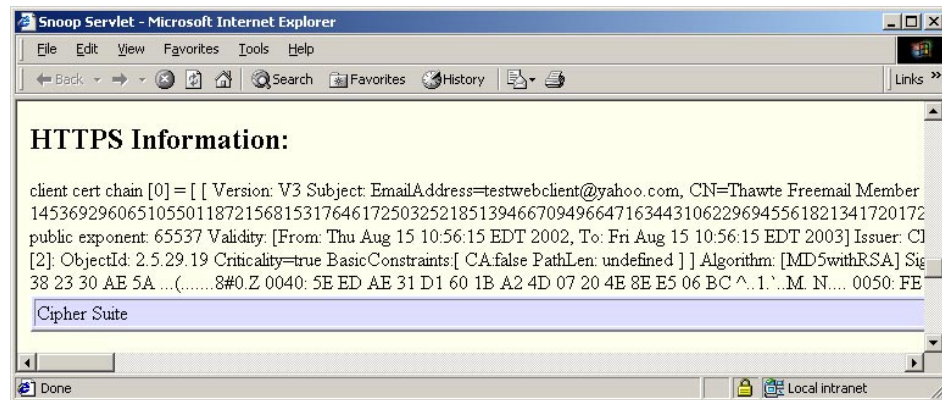
Figure 10-40   Response from Snoop using HTTPS and providing a client certificate

In Figure 10-40 above, the personal certificate installed in Microsoft Internet Explorer has successfully been passed to WebSphere. In the case that a client fails to pass a certificate, WebSphere will only return the Cipher suite specification as used in the HTTPS connections. If this occures, check that the client browser has a valid certificate installed and that your Web server is set to request client certificates.

A correctly configured implementation will prompt the client Web browser user to select a personal certificate when accessing a protected resource if the SSLClientAuth directive is set.

Open a new browser, if you have your browser open, then close it and open a new one. You will have to start the test with a completly new session. Go to the URL inyour browser: `http://<your_server/itsobank`, then follow the Customer Transfer link. The browser will present you will warn you that it was required to present the certificate; in case you have more than one certificate installed in your browser, then you wil have to select the right one for the Web site. Once you passed the certifiacte presentation part, you should be able to access the customer transfer page. In case the configuration is wrong or the certifiacte is not the right one you should get an Authorization error message.

You can follow the operation of the authentication if you have the tracing enabled for security. You should be able to find something similar in your trace.log file as the following example.

Example 10-5   trace.log

. . .

```
[10/14/02 18:35:21:764 EDT] 4461d823  > UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        mapCertificate
[10/14/02 18:35:21:764 EDT] 4461d823  > UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        search
[10/14/02 18:35:21:764 EDT] 4461d823  d UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        DN: o=itso
[10/14/02 18:35:21:764 EDT] 4461d823  d UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        Search scope: 2
[10/14/02 18:35:21:764 EDT] 4461d823  d UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        Filter: uniqueIdentifier=EmailAddress=testwebclient01@yahoo.com,
CN=Thawte Freemail Member
...
[10/14/02 18:35:21:774 EDT] 4461d823  < UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        search
[10/14/02 18:35:21:774 EDT] 4461d823  < UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        mapCertificate
[10/14/02 18:35:21:774 EDT] 4461d823  < UOW=
source=com.ibm.ws.security.registry.UserRegistryImpl org=IBM prod=WebSphere
component=Application Server
        mapCertificate parm1=cn=accountant01,o=itso
[10/14/02 18:35:21:774 EDT] 4461d823  d UOW=
source=com.ibm.ws.security.server.SecurityServerImpl org=IBM prod=WebSphere
component=Application Server
        Credential is a Certificate Credential. The user is =
cn=accountant01,o=itso
[10/14/02 18:35:21:774 EDT] 4461d823  > UOW=
source=com.ibm.ws.security.registry.UserRegistryImpl org=IBM prod=WebSphere
component=Application Server
        createCredential parm1=cn=accountant01,o=itso
...
```

Using the exact Distinguished Name

Using the Disinguished Name (DN) from the certificate to lookup the user means that the directory structure where the user can be found has to match with the DN. For example, the user DN is cn=user01,ou=ITSO,o=IBM,c=US; then the user01 has to be under the ITSO organizational unit (ou), IBM organization, US country (c) in this order.

This section provides information on how to use the exact distinguished name from the certificate to map the user to an LDAP user entry.

Acquiring a personal certificate

In this example we will use the Java keytool utility from sun to create the certifiacte request and then import it into a JKS keystore. Using the Java keytool, gives the flexibility to submit a request with the DN of our choice; IBM's ikeyman utility does not provide this flexibility at this moment.

Follow the steps below to acquire a new certificate from a CA.

1. Create the keystore for the user manager.

   ```
   keytool -genkey -keyalg RSA -dname "cn=manager01,o=itso" -alias manager01
   -keypass password -keystore testkeyring.jks -storepass password
   ```

2. Create a certificate request for the user.

   ```
   keytool -v -certreq -alias manager01 -file managerReq.csr -keypass password
   -keystore testkeyring.jks -storepass password
   ```

3. Send the request to the CA. This step requires some additional steps from the reader, where the *managerReq.csr* certificate request has to be sent to the CA for signing.

4. Get the CA public certificate. This step is a bit more complex, the user has to acquire the CA public certificate and import it into the keystore. The public certificate is either available as part of the Java Development Kit, or can be downloaded from the CA's Web site.

   ```
   keytool -import -alias "Trusted CA Certificate" -file CACert.cer -keystore
   testkeyring.jks -storepass password
   ```

5. Pick-up the certificate from the CA. This is again a bit more complex process, where the user has to get the signed certificate from the CA, save it in a simple text file under the name of *managerRespCert.arm*.

   ```
   keytool -import -trustcacerts -alias manager -file managerRespCert.arm
   -keystore testkeyring.jks -storepass password
   ```

6. You can use the JKS ikeyman tool to export the certificate in PKCS#12 format, in order to import it into the Web browser of your choice. When the Web site requires the user to present the certificate the user can pick the right one from the browser's keystore.

Configuring WebSphere to use exact DN mapping

The following steps will show you how to configure WebSphere Application Server to use Exact Distinguished Name (DN) mapping.

1. Login to the WebSphere Administration Console.

2. Select **Security -> User Registries -> LDAP**.

3. Select the **Advanced LDAP Settings** on the LDAP page.

4. Set the Certificate Map Mode to **EXACT_DN** on Configuration panel.

5. Make sure that the Certificate Filter field is empty.

6. Click **OK**, save the configuration for WebSphere.

7. Stop and start the server to make the changes available.

For testing use the same steps described before with the certificate filter option at Section , "Testing the client side certificate" on page 294.

You can follow the operation of the authentication if you have the tracing enabled for security. You should be able to find something similar in your trace.log file as the following example.

Example 10-6   trace.log

```
...
[10/14/02 19:39:38:318 EDT] 7a376025  > UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
          mapCertificate
[10/14/02 19:39:38:318 EDT] 7a376025  > UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
          search
[10/14/02 19:39:38:328 EDT] 7a376025  d UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
          DN: CN=manager01, O=itso
...
[10/14/02 19:39:38:348 EDT] 7a376025  d UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
          securityName = parm1=CN=manager01, O=itso
[10/14/02 19:39:38:348 EDT] 7a376025  d UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
          attributes = parm1={uid=uid: manager01, objectclass=objectclass:
inetOrgPerson, ePerson, organizationalPerson, person, top, cn=cn: Joe,
manager01}
```

```
[10/14/02 19:39:38:348 EDT] 7a376025  d UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        userName = parm1=manager01
[10/14/02 19:39:38:348 EDT] 7a376025  < UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        getUserDisplayName
[10/14/02 19:39:38:348 EDT] 7a376025  > UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        getUniqueGroupIds parm1=CN=manager01, O=itso
[10/14/02 19:39:38:348 EDT] 7a376025  > UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        getGroupsForUser
[10/14/02 19:39:38:348 EDT] 7a376025  d UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        filter = parm1=(|(&(objectclass=groupofnames)(member=CN=manager01,
O=itso))(&(objectclass=accessgroup)(member=CN=manager01,
O=itso))(&(objectclass=groupofuniquenames)(uniquemember=CN=manager01, O=itso)))
[10/14/02 19:39:38:348 EDT] 7a376025  > UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        search
[10/14/02 19:39:38:348 EDT] 7a376025  d UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        DN: o=itso
[10/14/02 19:39:38:348 EDT] 7a376025  d UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        Search scope: 2
[10/14/02 19:39:38:348 EDT] 7a376025  d UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        Filter: (|(&(objectclass=groupofnames)(member=CN=manager01,
O=itso))(&(objectclass=accessgroup)(member=CN=manager01,
O=itso))(&(objectclass=groupofuniquenames)(uniquemember=CN=manager01, O=itso)))
...
[10/14/02 19:39:38:348 EDT] 7a376025  < UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        search
[10/14/02 19:39:38:358 EDT] 7a376025  d UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        Number of groups returned = 1
```

```
[10/14/02 19:39:38:358 EDT] 7a376025  < UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        getGroupsForUser
[10/14/02 19:39:38:358 EDT] 7a376025  < UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        getUniqueGroupIds
[10/14/02 19:39:38:358 EDT] 7a376025  < UOW=
source=com.ibm.ws.security.registry.ldap.LdapRegistryImpl org=IBM
prod=WebSphere component=Application Server
        createCredential parm1=CN=manager01, O=itso
...
```

10.11 SSL between the Web server and WebSphere

This section documents the configuration necessary to instantiate a secure
connection between the Web server plugin 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 will cover the configuration
for the IBM HTTP Server 1.3.24, however the Web server related configuration in
this situation is not specific to any Web server.

Set authentication mechanism as client-cert.

The following steps are mandatory for generating the certificates for SSL
communicationbetween the two differing peers.

1. Create a self-signed certificate for the Web server plug-in

2. Create a self-signed certificate for the WebSphere embedded HTTP Server
   (Web Container)

3. Exchange the public keys between the two peers.

4. Modify the Web server plugin-cfg.xml file to use SSL/HTTPS.

5. Modify the WebSphere embedded HTTP Server (Web Container) to use
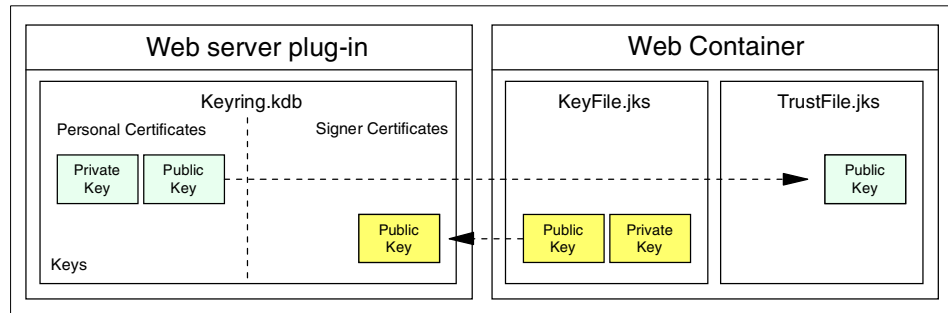   SSL/HTTPS.

Figure 10-41   Certificates

Figure 10-41 illustrates the exchange of the public certificate keys associated with each peer participating in the secure SSL communication.

Generating a self-signed certificate for the Web server plug-in

The Web server plug-in requires a keyring to store its own private and public keys and to store the public certificate from the Web container's keyfile.

The following steps will guide you through the process of generating a self-signed certificate for the Web server plug-in.

1. Create a suitable directory on the Web server host for storing the keyring file referenced by the plug-in and associated files; for example: `<IHS_root>/ssl`.

2. Launch the IBM ikeyman tool that ships as part of GSKit and supports the CMS key database format. This version of the ikeyman tool comes with the IBM HTTP Server. Note that it is not the version of ikeyman that comes with the WebSphere Application Server V5.

3. From the ikeyman menu, select: **Key Database File -> New**.

4. Set the following settings then click OK when you are done:

   Key database file: `CMS Key Database File`

   File name: `WASplugin.kdb`

   Location: `c:\IBMHttpServer\conf\keys\` (or the directory of you choice)

5. At the password prompt, enter the password of your choice; for our example use: `password`. Set the Stash the password to a file box, and the password will be saved to the stash file, so the plug-in can use the password to gain access to the certificates contained in the key database.

6. As we are only going to be implementing a peer-to-peer SSL connection between the Web server plug-in and the embedded HTTP server of any given Web container, we are not concerned with the signer certificates of the

publicly circulating root certificate authorities (CAs). In this case, optionally delete all of the CA trusted signer certificates.

7. From the ikeyman menu select: **Create -> New Self-Signed Certificate** to create a new self-signed certificate key pair. The following options then need to be specified; you may choose to complete all of the remaining fields for the sake of completeness:

   Key Label: `WASplugin`

   Version: `X509 V3`

   Key Size: `1024`

   Common Name: `websrv01.itso.ibm.com`

   Organization: `IBM`

   Country: `US`

   Validity Period: `365`

   Click **OK** when you are finished.

8. Extract the public self-signed certificate key, as this will be used later by the embedded HTTP server peer to authenticate connections originating from the plug-in.

9. Select **Personal Certificates** in the drop-down menu and select the **WASplugin** certificate that just was created.

10. Click the Extract Certificate button, ensuring that WASplugin remains selected. Extract the certificate to a file:

    Data type: `Base64-encoded ASCII data`

    Certificate file name: `WASpluginPubCert.arm`

    Location: `c:\IBMHttpServer\conf\keys` (or the directory of your choice)

    Click **OK** when you are finished.

11. Close the key database and quit ikeyman when you are finished.

Generating a self-signed certificate for the Web Container

The following steps will show how to generate a self-signed certificate for the WebSphere Web Container.

1. Launch the IBM JKS capable ikeyman version, that ships under the WebSphere bin directory. On Windows systems start it with the `ikeyman.bat` command, on UNIX systems run the `ikeyman.sh` script.

2. From the ikeyman menu select **Key Database File -> New**.

3. Set the following settings, then click OK when you are done:

   Key database file: `JKS`

File name: `WASWebContainer.jks`

Location: `c:\WebSphere\Appserver\etc\` (or the directory of your choice)

4. At the password prompt window, enter the password of your choice, for this sample use: `password`.

5. Optionally delete all the public Certificate Authority (CA) certificates under the Signer Certificates.

6. From the ikeyman menu select: **Create -> New Self-Signed Certificate**. Specify the fields with your values, the followings were used fo this samle:

    Key Label: `WASWebContainer`

    Version: `X509 V3`

    Key Size: `1024`

    Common Name: `wassrv01.itso.ibm.com`

    Organization: `IBM`

    Country: `US`

    Validity Period: `365`

    Click **OK** when you are finished.

7. Extract the public self-signed certificate key, as it will be used later by the Web server plug-in peer to authenticate connctions originating from the embedded HTTP server in WebSphere.

8. Select **Personal Certificates** from the drop-down list, then select the **WASWebContainer** certificate that was just created.

9. click the **Extract Certificate** button, ensuring that the **WASWebContainer** remains selected. Extract the certificate to a file:

    Data type: `Base64-encoded ASCII data`

    Certificate file name: `WASWebContainerPubCert.arm`

    Location: `c:\WebSphere\Appserver\etc`

    Click **OK** when you are finished.

10. Close the database and quit ikeyman when you are finished.

Exchanging public certificates

The following two sections will describe how to exchange certificates between the Web Container keystore and the Web server plug-in keyfile.

In order to import the certificates into the keystores as described in the next two sections, you will have to copy over the two certificates, the extracted .arm files, to both machines, Web server, WebSphere server.

1. Copy the WASpluginPubCert.arm from the Web server machine to the WebSphere machine. The source directory in our case is c:\ihs\conf\keys, while the destination is: c:`\WebSphere\Appserver\etc`.

2. Copy the WASWebContainerPubCert.arm from the WebSphere machine to the Web server machine. The source directory in our case is c:\was\etc, while the destination is: c:\IBMHttpServer\conf\keys.

Importing the certificate into the Web server plug-in keyfile

1. On the Web server machine launch the ikeyman utility that supports the CMS key database format.

2. From the ikeyman menu select **Key Database File -> Open** and select the previously created key database file: **WASplugin.kdb**.

3. At the password prompt window, enter the password then click **OK**.

4. Select **Signer Certificates** from the drop-down list, then click the **Add** button. This will allow you to import the public certificate previously extracted from the embedded HTTP server/Web Container keystore.

   Data type: `Base64-encoded ASCII data`

   Certificate file name: `WASWebContainerPubCert.arm`

   Location: `c:\WebSphere\Appserver\etc\`

   Click **OK** when you are finished.

5. You will be prompted for a label name by which the trusted signer public certificate will be known. Enter a label for the certificate: **WASWebContainer**.

6. Close the key database and quit ikeyman when you are finished.

Importing the certificate into the Web Container keystore

1. On the WebSphere machine, launch the IBM JKS capable ikeyman version that ships under the WebSphere bin directory.

2. From the ikeyman menu, select: **Key Database File -> Open** and select the previously created **WASWebContainer.jks** file.

3. At the password prompt, enter the password for the keyfile then click **OK**.

4. Select **Signer Certificates** in the drop-down list and click the **Add** button. This will allow you to import the public certificate previously extracted from the Web server plug-in keyfile.

   Data type: `Base64-encoded ASCII data`

   Certificate file name: `WASpluginPubCert.arm`

   Location: `c:\WebSphere\Appserver\etc\`

   Click **OK** when you are finished.

5. You will be prompted for a label name by which the trusted signer public certificate will be known. Enter a label for the certificate: **WASplugin**.

6. Close the key database and quit ikeyman when you are finished.

Modifying the Web server plug-in file

The plug-in config file must be modified to reference the plug-in keyring and the password stash file. This allows the transport protocol to be changed from HTTP to HTTPS, using the certificates stored in the keyring.

A standard non-secure HTTP connection in the configuration looks like the following:

```
<Transport Hostname="wassrv01" Port="9080" Protocol="http"/>
```

The same entry, but secured, looks like the following:

```
<Transport Hostname="wassrv01" Port="9080" Protocol="https">
    <Property name="keyring"
value="c:\IBMHttpServer\conf\keys\WASplugin.kdb"/>
    <Property name="stashfile"
value="c:\IBMHttpServer\conf\keys\WASplugin.sth"/>
</Transport>
```

> **Note:** the Transport XML tag has a body and a closing tag, make sure you remove the slash '/' from the end of the opening tag.

Transport protocol and SSL key properties can be specified for each transport. In the previous example the simple HTTP transport had been secured. However it does not make much sense, since the communication from the client to the Web server and the plug-in is not secured. The secure port for the WebSphere Application Server 9433 is already defined in the plugin configuration, and it is configured to use SSL/HTTPS.

It might be useful for production environment to replace the original plugin-key.kdb file with your own key file for the secure transport definition, port 9443.

Modifying the Web Container to support SSL

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

The following steps document the required Web Container modifications.

1. Start the WebSphere Administration Console then after login, select: **Security -> SSL.**

2. Click **New** to create a new entry in the repertoire. Provide the following values to fill out the form:

Alias: `Web Container SSL`

Key File Name: `c:\WebSphere\Appserver\etc\WASWebContainer.jks`

Key File Password: `password`

Key File Format: `JKS`

Trust File Name: `c:\WebSphere\Appserver\etc\WASWebContainer.jks`

Trust File Password: `password`

Trust File Format: `JKS`

Client Authentication: `unchecked`

Security Level: `HIGH`

click **OK** when you have finished.

> **Note:** in case you want mutual SSL between the two parties, check the Client Authentication check-box.

3. **Save** the configuration in the WebSphere Administration Console.

4. Select the **Servers -> Application Servers**, then select the server you want to work with, in this case: **server1**.

5. Select the **Web Container** under the server.

6. Select **HTTP Transport** under the Web Container.

7. Select the entry for the transfer you want to secure, click the item under the Host column. Select the ***** (asterisk) in this case in the line where 9080 is the Port.

8. On the configuration panel, check-in the **Enable SSL** box, and select the desired SSL entry from the repertoire from the SSL drop-down list in our case the **Web Container SSL**.

9. click **OK**, then save the configuration for WebSphere.

Testing the secure connection

To test the secure connection use your favorite Web browser and access a Web application on WebSphere Application Server using the port 9080, for example: `https://wassrv01.itso.ibm.com:9080/itsobank`.

Make sure you use the https protocol, if not, the returned page will look like the following.
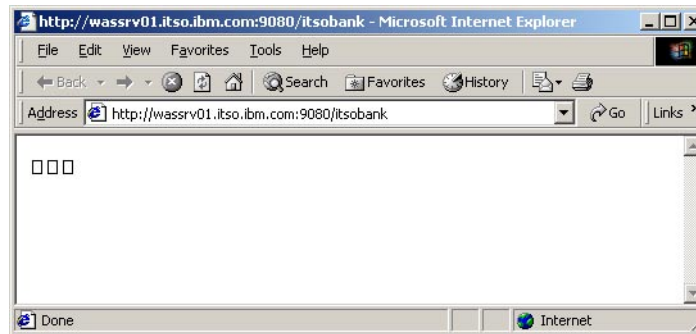
Figure 10-42   False HTTP response

In order to test the secured connection when client side certification is required, the right certificate with public and private key has to be imported into the browser.

1. On the Web server machine, launch the `ikeyman` utility that can handle the CMS key database file.

2. Open the keyfile for the plugin, in our example: `c:\IBMHttpServer\conf\keys\WASplugin.kdb`. Provide the password when it is prompted.

3. Selec the **WASplugin** certificate under the Personal Certificates, then click **Export**.

4. Save the certificate in PKCS12 format to a file, c:\IBMHttpServer\conf\keys\WASplugin.p12. Provide a password to secure the PKCS12 certificate file, then on the next panel select **Weak encryption (browser compatible)**.

5. Close the keyfile and quit ikeyman when you are done.

6. Copy the saved WASplugin.p12 file to the client machine where you want to access the WebSphere server from.

7. Import the PKCS12 file into your favorite browser. In Microsoft Internet Explorer, select **Tools -> Internet Options ...** from the menu. Switch to the **Content** tab then click **Certificates**. Import the WASplugin.p12 certificate by using the **Import...** button; provide the password for the file where it is necessary. The new certificate should appear under the personal tab. Close the certificates and the options dialog.

8. In the browser access the WebSphere application again, for example: `https://wassrv01.itso.ibm.com:9080/itsobank`.

9. The browser will ask, which personal certificate to use for the connection, select the certificate then continue the connection.

10.The Web page should come up with the right content.

Once the browser test with the direct WebSphere access was successful, test the connection through the Web server. Open a Web browser and access the Web application using the normal port settings (port 80), for example: `http://wassrv01.itso.ibm.com/itsobank`. The Web page should appear with the right content.

10.12 SSL between the Java client and WebSphere

SSL may be used to secure a connection between two ORBs. When a Java client invokes a method on a remote EJB, the client and server ORBs will communicate information in the clear. SSL can protect information being passed over the IIOP protocol in the same way that it protects information being passed over other protocols.

In order to establish an inter-ORB connection, WebSphere demands that the identity of the client is provided. This has no relation to the role-based security used to protect the J2EE application, although the identity passed during the SSL initialisation sequence can be used for authorisation purposes.

10.12.1 Creating the key stores

In order to secure the ORB communication you will need to create the key file and trust file pairs for the server and the client, then you also have to exchange the certificates between the two parties.

Follow the steps in Section 10.9, "Demo keyfile" on page 260 to create the key store, trust store file pairs. Use the following file and keylabel names and save them in a directiry where you will find them, for example: <WebSphere_root>\etc.

► Server key file: `ServerKeyFile.jks`

► Server certificate label: `ServerKey`

► Server trust file: `ServerTrustFile.jks`

► Client key file: `ClientKeyFile.jks`

► Client certificate label: `ClientKey`

► Client trust file: `ClientTrustFile.jks`

Exchange the certificates between the two parties, export the ServerKey from the ServerKeyFile.jks and import into the ClientTrustFile.jks; export the ClientKey from the ClientKeyFile.jks and import into the ServerTrustFile.jks.

10.12.2 Server side configuration

The Application Server must be configured to support SSL. An SSL configuration should exist that describes the type of key stores used to establish the secure connection and their location. Refer to Chapter 10.9.4, "Configuring WebSphere to use a key store" on page 274 for details on key stores.

> **Note:** The sas.server.props configuration file used in WebSphere Application Server, version 4 is no longer used in version 5. However, the file remains in the properties directory. The server security configuration is contained in a file called security.xml whose default location is <WebSphere_root>/config/cells/BaseApplicationServerCell.

Create a new SSL entry in the SSL Repertoire, following the steps in Section 10.8.1, "SSL configurations" on page 258 using the following values for the attributes:

SSL alias: `ORB SSL`

Key file: `C:\WebSphere\Appserver\etc\ServerKeyFile.jks`

Key file password: `password`

Trust file: `C:\WebSphere\Appserver\etc\ServerTrustFile.jks`

Trust file password: `password`

The authentication protocol must be configured to use the correct SSL settings.

1. Login to the WebSphere Admin console, select **Security -> Authentication Protocol -> CSI -> Authentication -> Inbound**.

2. Ensure that Basic Authentication is supported, at the very least. It is also valid to set Basic Authentication to Required.

> **Note:** When *required* is set to true for an attribute, where *supported* is also an option, the supported attribute will not be used by the server.
>
> It is true for every attributes within CSI also.

3. Client Certificate Authentication may be set to Supported or Never. A client certificate is not required in order to establish an SSL connection between the client and WebSphere.

4. Uncheck Identity Assertion since this option is not required.

5. Ensure there are no trusted servers listed.

6. Check the Stateful checkbox as this provides a useful increase in performance due to the fact that the client only needs to authenticate once per session.

7. Click **OK.**
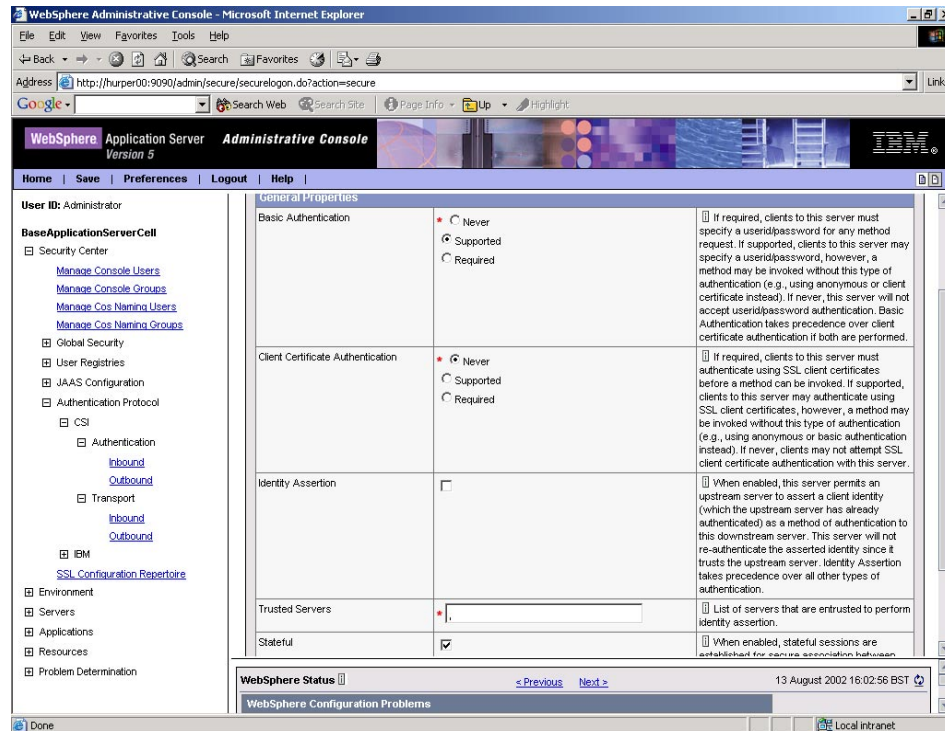
Figure 10-43   CSIv2 authentication configuration - redo

8. Select **Security -> Authentication Protocol -> CSI -> Transport -> Inbound**

9. Ensure that the transport is set to either SSL-Required or SSL-supported, in our case set it to **SSL-Required** in order to force SSL connection.

10. Select the pre-defined SSL configuration from the SSL Settings list, in our case select **ORB SSL**.

11. Click **OK.**
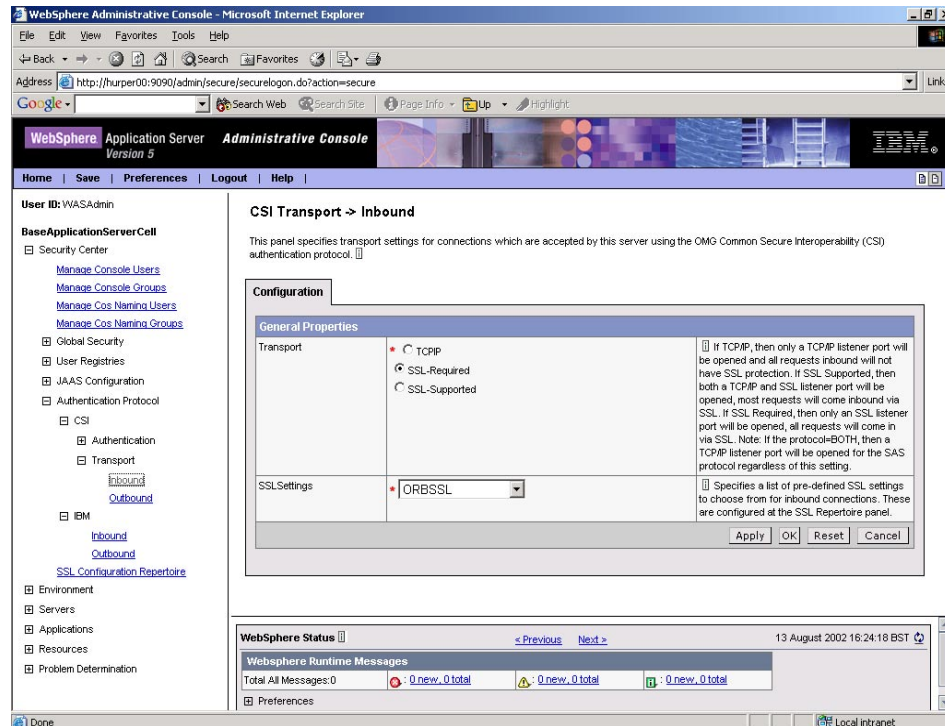
Figure 10-44   CSIv2 transport configuration - redo

This completes the configuration necessary for CSIv2-based connections. However, some Java client may not support CSIv2 and in this case WebSphere will need to provide backwards-compatibility with version 4 in the form of IBM's SAS protocol. Configuring this protocol to use SSL is relatively straight-forward.

12. Select **Security -> Authentication Protocol -> IBM -> Inbound.**

13. Select the appropriate SSL configuration from the SSL Settings list, in our case select **ORB SSL**.

14. Click **OK.**
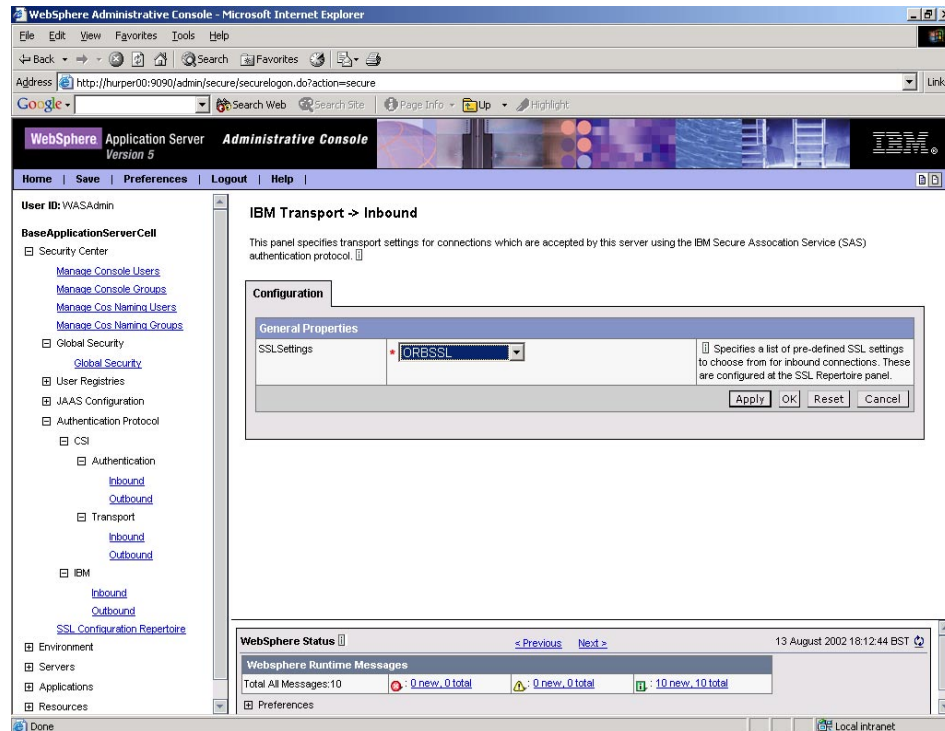
Figure 10-45   IBM SAS configuration - redo

15. The changes should be saved and WebSphere restarted.

Of course, it is necessary for Global Security to be enabled for SSL to operate and so ensure that this is the case.

10.12.3 Configuring the Java client

A J2EE application client environment provides the necessary libraries to connect via SSL and so no additional effort from the client application developer is required. Configuration is provided in the sas.client.props file which is located in WebSphere's properties directory, by default. For details regarding the options available in this file, consult Chapter 6.3, "Configuring the Java client" on page 103.

Before commencing configuration of the Java client, it is recommended to backup the current sas.client.props file.

To allow for a secure connection to WebSphere, the following settings are necessary.

Table 10-4   sas.client.props configuration

Property	Value
com.ibm.CORBA.securityEnabled	true
com.ibm.ssl.protocol	SSL
com.ibm.CSI.performTransportAssocSSLTLSSupported	true
com.ibm.CSI.performTransportAssocSSLTLSRequired	true
com.ibm.CSI.performMessageIntegritySupported	true
com.ibm.CSI.performMessageConfidentialitySupported	true

Once the sas.client.props file has been updated, it should be saved to the filesystem.

Testing the configuration involves starting a Java client with the launchclient tool. Information regarding the launchclient tool can be found in Chapter 6.5, "J2EE application client" on page 121.

An addtion requirement is that the client provides a certificate in order to establish its identity, rather than a user name and password. Java clients can be configured to provide a certificate and in this case, several additional steps are necessary.

The sas.client.props file must be updated to reflect the addition of the client trust store.

Table 10-5   sas.client.props configuration

Property	Value
com.ibm.ssl.keyStoreType	JKS (if ikeyman is used)
com.ibm.ssl.keyStore	<location of key file>
com.ibm.ssl.keyStorePassword	<password for key file>
com.ibm.ssl.trustStoreType	JKS
com.ibm.ssl.trustStore	<location of trust file>
com.ibm.ssl.trustStorePassword	<password for trust file>
com.ibm.ssl.protocol	SSLv3
com.ibm.CSI.performTLClientAuthenticationSupported	true

Property	Value
com.ibm.CSI.performClientAuthenticationRequired	false (to ensure that the client certificate is being used)

It is possible to configure keystores each for CSIv2 and IBM SAS connections. In this case the com.ibm.ssl properties shown in the table above should be replaced with the following.

For CSIv2-based connections.

Table 10-6   CSIv2-specific sas.client.props configuration

Property	Value
com.ibm.ssl.csiv2.outbound.keyStore	<location of client CSIv2 key file>
com.ibm.ssl.csiv2.outbound.keyStorePassword	<password for client CSIv2 key file>
com.ibm.ssl.csiv2.outbound.keyStoreType	JKS
com.ibm.ssl.csiv2.outbound.trustStore	<location of client CSIv2 trust file>
com.ibm.ssl.csiv2.outbound.trustStorePassword	<password for client CSIv2 trust file>
com.ibm.ssl.csiv2.outbound.trustStoreType	JKS
com.ibm.ssl.csiv2.outbound.protocol	SSLv3

For IBM SAS-based connections.

Table 10-7   IBM SAS-specific sas.client.props configuration

Property	Value
com.ibm.ssl.sas.outbound.keyStore	<location of client SAS key file>
com.ibm.ssl.sas.outbound.keyStorePassword	<password for client SAS key file>
com.ibm.ssl.sas.outbound.keyStoreType	JKS
com.ibm.ssl.sas.outbound.trustStore	<location of client SAS trust file>

Property	Value
`com.ibm.ssl.sas.outbound.trustStorePassword`	<password for client SAS trust file>
`com.ibm.ssl.sas.outbound.trustStoreType`	JKS
`com.ibm.ssl.sas.outbound.protocol`	SSLv3

Once SSL is configured, use the launchclient tool provided with WebSphere to test the connection. It may be feasible to run a packet monitoring tool to be sure that the information passing from client to server is, in fact, encrypted. Only the server and client certificates should be sent in the clear and then only during the initialisation stage. In any case, these certificates are considered to be viewable by the public in general as they do not contain any private information.

Should an error occur, the likelihood is that a Java exception trace will appear in the client console. Often the errors refer to CORBA problems which is the underlying marshalling mechanism with which the ORBs operate. Most CORBA exceptions are difficult to interpret due to their somewhat terse messages. Tracing can also provide a useful insight to the events that led up to the error.

10.13 Connecting to directory servers (LDAP)

This section will discuss the LDAP User Registry configuration for the WebSphere Application Server. The user registry we used to show the configuration steps is the IBM SecureWay Directory Server V3.2.2. This section will tell you how to configure your LDAP server for this sample, how to create a sample user and a sample group entry in the directory. Tha major part of the section is to give an example how to configure WebSphere to use a given LDAP server over a normal LDAP connection then using SSL for LDAP (LDAPS).

For other LDAP servers refer to Appendix B, "LDAP configurations" on page 457.

10.13.1 IBM SecureWay Directory Server V3.2.2

The following detailed configuration will show, how to configure WebSphere Application Server V5 to use the IBM Secureway Directory Server V3.2.2. There are two scenarios, where the second build upon the first one;

► The first scenario covers the basic LDAP configuration with WebSphere Application Server.

► The second scenario covers how to enable the connection to use SSL for LDAP (LDAPS) providing security to WebSphere LDAP communication.

Before going for securing the connection between WebSphere and LDAP communication using SSL, we recommend that we first configure LDAP configuration for WebSphere.

Configuring basic LDAP connection

The following steps will show a basic configuration for WebSphere Application Server V5 to use IBM SecureWay Directory Server as the user registry.

Configuring the IBM SecureWay Directory Server

Once the installation and basic configuration for the directory server is finished, proceed to add new data entries into the directory. The following steps will guide you through the basic configuration of IBM SecureWay Directory Server.

Before you can add entries to the database, it is necessary to define a suffix for that directory. A suffix is the starting point in the directory and specifies the Distinguished Name (DN) for the root of that tree. The LDAP server must have at least one suffix defined and can have multiple suffixes. Each entry added to the directory contains in their fully Distinguished Name (DN) a suffix that matches one of the server's suffixes defined on the server.

To define a valid suffix, it is possible to use the X.500 naming structure that will set the root of the directory to a specific organization in a specific country or to a specific organization and organizational unit:

```
o=ibm,c=us
```

where "o" represents Organization and "c" represents Country, and

```
ou=raleigh,o=ibm
```

where "ou" represents Organizational Unit and "o" represents Organization.

It is also possible to use the DNS naming model by using the *domainComponent* attribute:

```
dc=ibm.com
```

where "dc" represents a domain component, for example:

```
dc=itso,dc=ral,dc=ibm,dc=com
```

To add a suffix in the directory, follow these steps:

1. Open a Web browser, then access the URL for the Server Administration:
   `http://<your server>/ldap` .

2. Once logged in as an administrator, click the Add Suffixes link at the top of the page or expand the **Settings -> Suffixes** folder then click **Add** Suffixes.
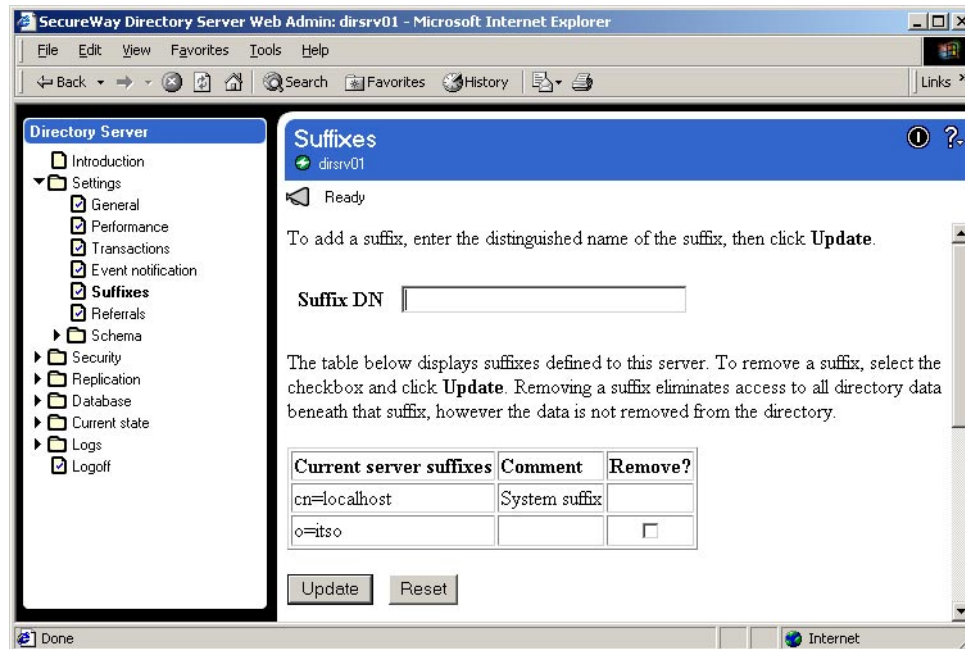
Figure 10-46   Create a Siffixes in IBM SecureWay Directory Server Administration

For our example, using the X.500 methodology, we set the following suffix: o=itso. Click **Update**, and the Suffix table list will be updated with the new suffix. At this point, it contains no data.

3. Restart the server by clicking the **Restart the Server** link on the top of the page so that the changes may take effect.

Create a user entry

To add new user entries in the directory, use the Directory Management Tool tool Follow the steps below to add a new user:

1. Open the Directory Management Tool by clicking **Start -> Programs -> IBM SecureWay Directory -> Directory Management Tool**.

2. Click the **Add Server** button. The Add Server fields are displayed. Type in the Server Name and Port and then select **Simple** in the Authentication Type. Log on as the cn=root user and click **OK**.

3. A Warning message box will be displayed, notifying you that the suffix created previously (itso) does not contain any data. Click **OK**.

> **Note:** If an authenticated user is not introduced during the Add Server process, you will only have anonymous privileges; this means that it is only possible for you to browse the directory, but not to modify it. To authenticate yourself, select **Server -> Rebind** from the menu in the left-hand pane.

4. To add a new entry in the directory select the ldap://<your_ldap_server>:389 itam and click the **Add** button in the toolbar. Select Organization as the Entry Type and `o=itso` as the Entry RDN (Relative Distinguished Name), as shown in Figure 10-47.
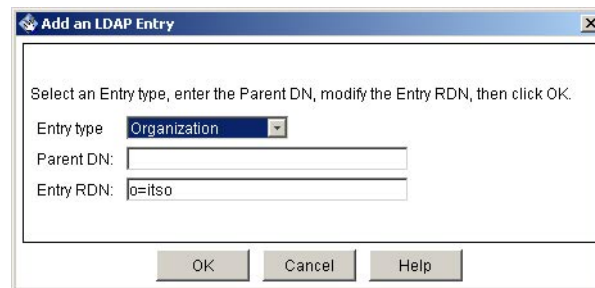
Figure 10-47   Adding an organization

5. Click **OK** and a new window appears for setting attributes to the new RDN entry. All the fields marked in bold style are mandatory, in this case it is only the "o" filed, which already has a value assigned.

6. Click **Add**. The new organization entry should appear in the directory tree after clicking **Directory -> Refresh** Tree as shown in Figure 10-48.
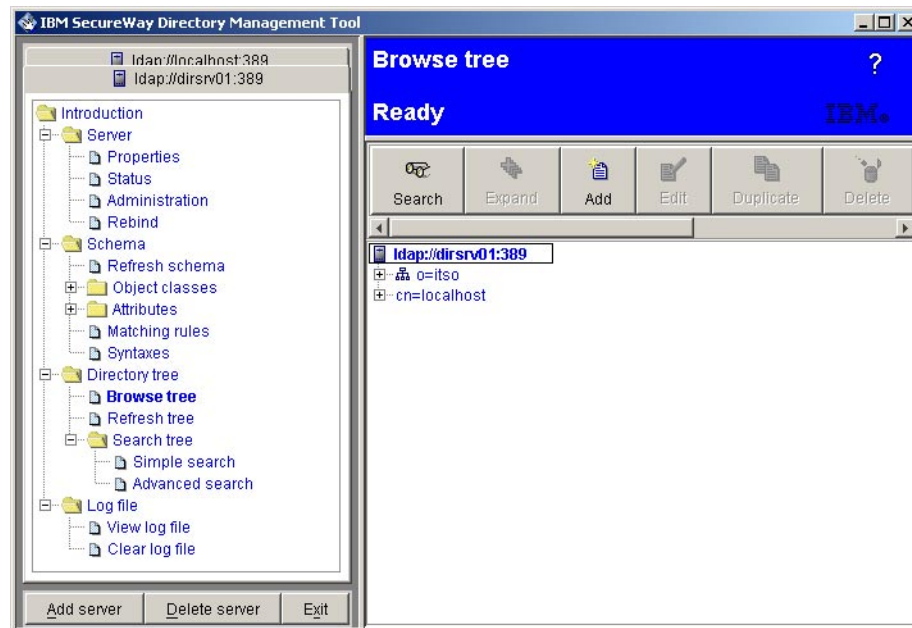
Figure 10-48   New Organization in the Directory after refreshing

7. Now, proceed to add new users to the new organization created previously. We are going to add a new administrative user in order to set a Security Server ID in the WebSphere Global Security settings, for use as the user ID under which the server runs for security purposes.

    Select the country o=itso and click the **Add** button on the tool bar. Select **User** as the Entry Type and type `cn=wasadmin` as the Entry RDN.
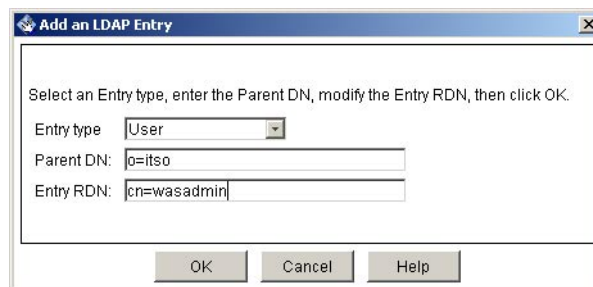
Figure 10-49   Adding a new user

8. Click **OK**; a new window appears for setting attributes to the new RDN entry, as shown in Figure 10-50.

The Distinguished Name (DN) is the fully qualified name for the user. The default DN is the Parent DN plus the Entry RDN. Enter the Last Name (required for adding new users) and a password further below. Include any other information in the Business and Personal tabs.

Figure 10-50   Setting attributes to the new user

9. Switch to the **Other** tab and supply the UID to allow the user to authenticate himself with this value instead of using the full DN, type in: `wasadmin`.

10. Click **Add**. The new user entry should appear in the directory tree beneath the organization entry (o=itso) after clicking **Directory -> Refresh Tree**.

Create a group entity

To add new groups in the directory, it is possible to use the DMT (Directory Management Tool) or provide the data in an LDIF (LDAP Data Interchange Format) file. To add new groups, follow these next steps:

1. Select the organization **o=itso** and click the **Add** button in the toolbar.

2. Select **Group** as the Entry Type and type `cn=admingrp` as the Entry RDN as seen in Figure 10-51; then click **OK**.
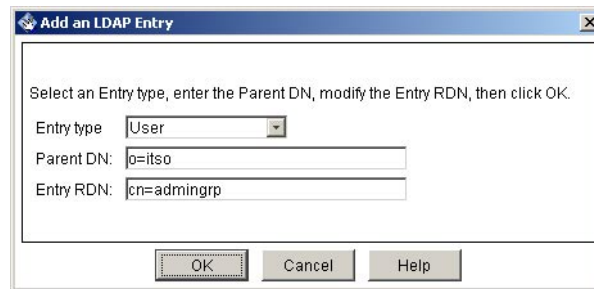
Figure 10-51   Adding a new Group

3. The next window is used for setting attributes for the new RDN entry, as shown in Figure 10-52.

Figure 10-52   Setting Groups attributes

The member (Group member) field is mandatory; assign the following group member to the group: `cn=wasadmin, o=itso`.

4. After creating all the users and groups for the application and clicking **Directory -> Refresh Tree** the directory structure should appear as it is shown for the ITSOBank applicaiton on the picture below.
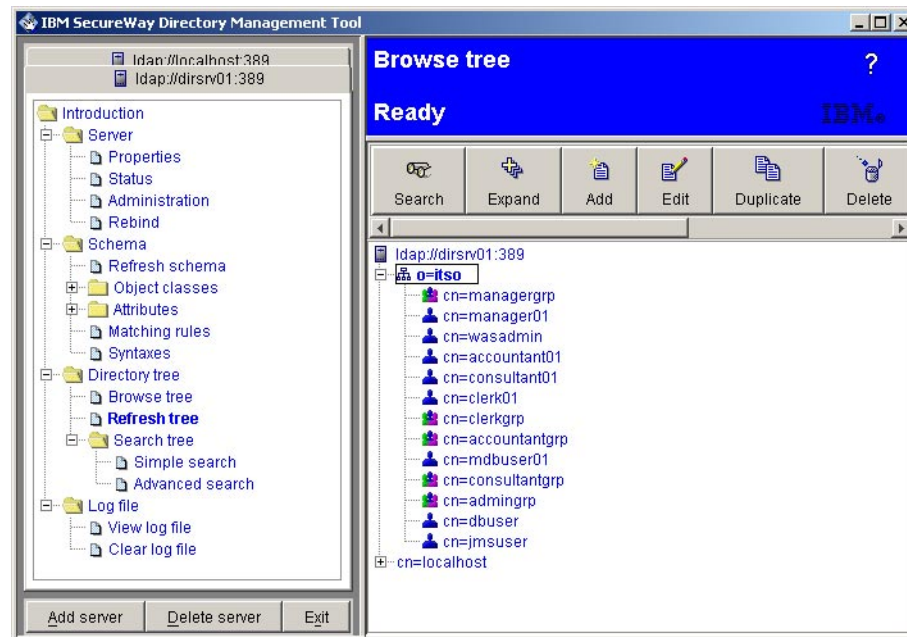
Figure 10-53   LDAP directory with users and groups for the ITSOBank sample

Mass data load

As mentioned before, it is possible to add users and groups using an LDIF file, a standard format for representing LDAP entries in text form. This will be useful when the number of entries to add is very large. An example of an LDIF file is shown below.

Example 10-7   Sample LDIF File (code snippet)

```
version: 1

dn: o=itso
objectclass: top
objectclass: organization
o: itso

dn: cn=wasadmin,o=itso
objectclass: top
objectclass: organizationalPerson
objectclass: inetOrgPerson
objectclass: person
objectclass: ePerson
uid: wasadmin
userpassword: {iMASK}>1E6ViLtjfRnaIvsM9gJYbK5u3X/FqvrAAKQuhmmr47OFebq5EJyZ76u5
```

```
ayIfX+qeEFcWEpDWEHEWCEGcOsRADuWtbzl5FxHpt3uoPT88vOG8gGo/OgmHkqOP88xU9NmaAt+h
GKqCqr76TeJcmkUwejjnPb8pAoIYCq<
sn: wasadmin
cn: wasadmin

dn: cn=admingrp,o=itso
objectclass: groupOfNames
objectclass: top
cn: admingrp
member: cn=wasadmin,o=itso
```

To import the file, perform the following steps:

1. Open the browser and access the URL for Server Administration using the format `http://<your server>/ldap`. Once logged as a root , check that the Directory server is running.

2. Select **Database -> Import LDIF**.

3. Enter the name of the LDIF file on the LDAP server from which you want to import directory data, then click **Import**.

4. Wait until a message appears indicating that the entries have been successfully added.

5. Use the DMT tool to verify the new entries.

Configuring WebSphere to use LDAP User registry

In order to use the LDAP directory of your choice as the user registry, WebSphere has to be configured in a proper way.

1. Launch the Administrative Console for WebSphere.

2. Select **Security -> User Registries -> LDAP**, this page will provide the settings for the LDAP configuration.

3. Provide all the information shown on Figure 10-54 on page 324 according to your system settings.

Figure 10-54   WebSphere LDAP Configuration panel

4. Click **Apply** to keep the settings.

5. For special LDAP search settingd click the Advanced LDAP Settings link on the bottom of the page to get to the following page, Figure 10-55 on page 325.

Figure 10-55   LDAP Advanced settings Configuration panel

6. The settings here are correct for the chosen SecureWay Directory Server. The settings will change accordingly whenever the type of the directory is changed on the main LDAP configuration page. You can also configure your own custom LDAP search strings on this page.

7. Once finished with the LDAP configuration save the settings to make it available for WebSphere.

8. You will need to restart the application server in order to make the changes effective.

> **Note:** Even if global security is not enabled you can set the user registry to LDAP for WebSphere. There are cases when a user registry is needed even without security, it will let you to do role mapping during application deployment, or use LDAP users for J2C authentication entries for the datasource adapter.

Configuring secure LDAP (LDAPS) connection

This section requires to configure LDAP connection for WebSphere Application Server V5 by following the previous steps from Section , "Configuring basic LDAP connection" on page 316.

Creating the certificates for SSL

This section provides information for the keyring settings needed for the secure LDAP connection over SSL.

Creating a self-signed certificate for the SecureWay LDAP peer, follow the steps in Section 10.10.1, "Generating a digital certificate" on page 277.

- ► Use the following information for the new LDAP keyring file:

    Key database file: `CMS Key database file`

    File name: `LDAPKey.kdb`

    Location: `C:\LDAP\etc`

- ► Use the following information to create the LDAP key entry:

    Key label: `LDAP SSL`

    For the rest of the fields use your own settings according to your server and location.

- ► For extracting the certificate from the LDAP keyring file use the following details:

    Data type: `Base64-encoded ASCII data`

    Certificate file name: `SecurewayDAPCert.arm`

    Location: `C:\LDAP\etc`

Creating a Key database for the WebSphere LDAP SSL peer, follow the steps from Section 10.9.1, "Generating a self-signed certificate" on page 263.

- ► Use the following information for creating the new key database:

    Key Database File: `JKS`

    File Name: `WASLDAPKeyring.jks`

    Location: `C:\WebSphere\AppServer\etc`

- ► Use the following information to create a new self-signed certificate for LDAP:

    Key label: `LDAPSSL`

    For the rest of the fields use your own settings according to your server and location.

- ► For extracting the certificate, use the following information:

    Data type: `Base64-encoded ASCII data`

    Certificate file name: `WebSphereLDAPCert.arm`

    Location: `C:\WebSphere\AppServer\etc`

To exchange the certificates between the two keyrings follow the steps from Section , "Exchanging public certificates" on page 303. using the following details:

► LDAP certificate file `C:\LDAP\etc\SecurewayDAPCert.arm` has to be copied to the WebSphere server, then imported into the `C:\WebSphere\AppServer\etc\WASLDAPKeyring.jks` file.

► WebSphere certificate `C:\WebSphere\AppServer\etc\WebSphereLDAPCert.arm` has to be copied to the LDAP server, then imported into the `C:\LDAP\etc\LDAPKey.kdb` file.

Configuring the IBM SecureWay Directory Server
After successfully generating and exchanging the SecureWay public key, both the SecureWay Directory Server and WebSphere must be configured to support SSL. The assumption is made that you have previously installed and configured the SecureWay LDAP Directory for authenticating WebSphere users, albeit without SSL.
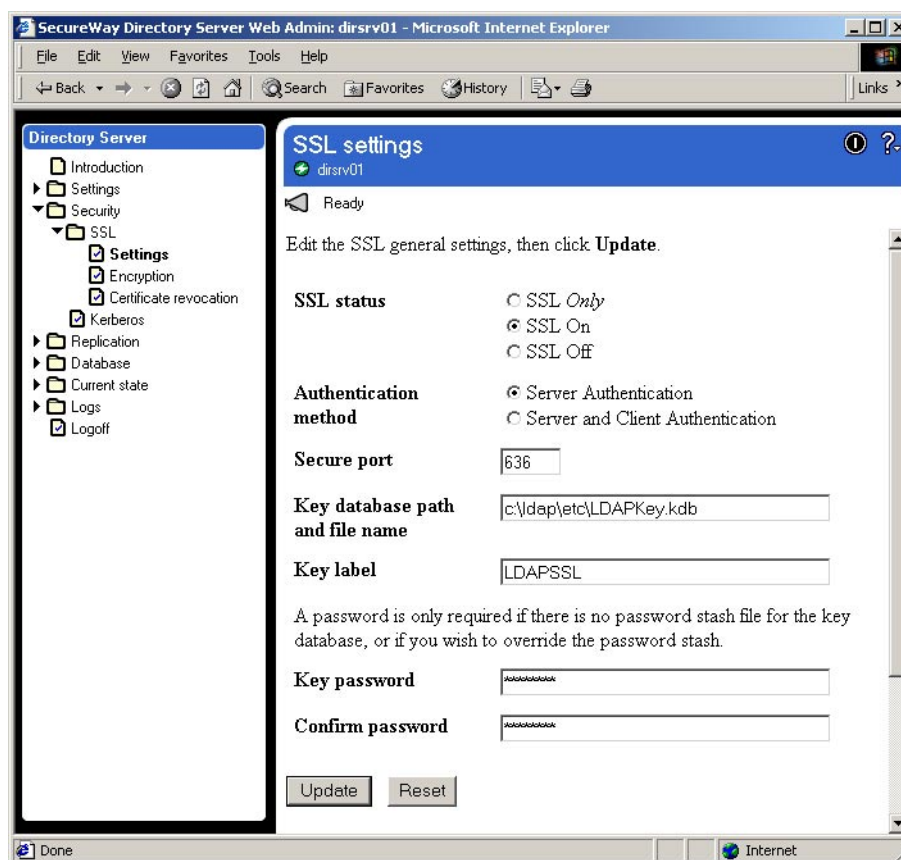
Figure 10-56   Configure SSL for SecureWay Directory Server

Complete the following steps to enable LDAP SSL communication.

1. Launch the SecureWay Web-based administration console in your chosen Web browser. Open the URL `http://<ldap_servername>/ldap`.

2. Complete the LDAP Administrator ID and Password fields when prompted to authenticate yourself to the Directory Server. Typically, the Distinguished Name (DN) cn=root is used here. However, any user with sufficient privileges can perform this task.

3. From the Directory Server topology tree in the left pane, select the **SSL Settings** tab found under the Security heading. Figure 10-56 on page 328 shows the corresponding SSL Settings window that will be displayed in the right-hand-side pane.

4. You must complete the following fields to enable LDAP communication over SSL:

SSL status: select **SSL On** or *SSL Only* if you wish to prevent non-SSL LDAP connections.

Authentication method: select **Server Authentication**. You may choose to select **Server** and **Client Authentication**, in which case you need to ensure that the public certificate key associated with any authenticating client is resident in your (LDAP) certificate key database.

Secure port: select **636** which is the Internet standard for secure LDAP communication. You may choose any port that is not in use. On Unix, only the root has access to the ports below 1024 by default.

Key database path and file name: specify the fully qualified file name of the CMS key database previously created. In our example, this is set to `C:/LDAP/config/SecureWayLDAP.kdb` . SecureWay does not support the Java Key Store (JKS) type certificate key database.

Key label: as a key database can contain multiple certificates, specify the label name of the certificate used for authenticating the LDAP Directory Server. In our example this is set to `LDAPSSL`.

Key password: specify the key database password if you did not generate a password stash file when creating the certificate key database originally. This password will be used by SecureWay to gain access to the certificate database.

5. Click the **Update** button when you have completed all of the above fields.

6. For the changes to be included into the runtime, you must stop and restart the LDAP Directory Server. Once restarted, you can check the status of the Directory by expanding the **Current State** and **Server Status** menus. If the Directory fails to start, check the Error logs. Unix users can also check that the Directory is listening for incoming SSL LDAP connections by using the `netstat -a` command and "grepping" for port 636.

If you are concerned with the level of SSL support offered by the SecureWay LDAP Directory Server, you can choose to restrict the permitted encryption algorithms. For example, you may decide that (40-bit) encryption is inadequate for your SSL implementation. In this case, the (40-bit) encryption method can be deselcted, as shown below in Example 10-57.
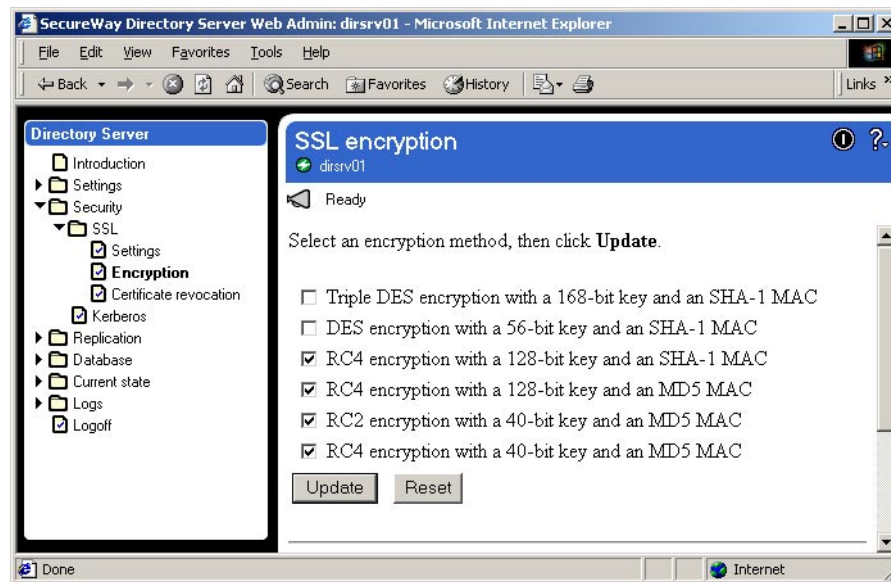
Figure 10-57   SSL Encryption settings for SecureWay Directory Server

Bear in mind that each SSL peer must support the same encryption method/cipher suite to be able to establish an encrypted session. The encryption methods supported by WebSphere are classified into three categories high , medium and low , and are configured via the Security level setting on the LDAP SSL Configuration window as shown in Figure 10-58 on page 331

Configuring WebSphere Application Server V5
The following steps will guide you through the basic configuration of WebSphere Application Server V5.

SSL Configuration
We need to configure the keyring for SSL we previously created in Section , "Creating the certificates for SSL" on page 326, using Administrative Console for WebSphere Application Server V5. Following are the steps:

Figure 10-58   Configuring SSL Certificate using Administrative Console

1. Click **Security -> SSL.**

2. Fill in the configuration values as follows:

   Key File Name: specify the fully qualified file name of the Java Key Store
   (JKS) key database previously created. In our example this is set to
   `C:\WebSphere\AppServer\etc\WASLDAPKeyRing.jks` .

   Key File Password: state the password used to protect the Java Key Store
   (JKS) certificate key database above.

   Key file format: ensure that **JKS** is selected.

Trust file name: potentially, you can set this to point at a second Java Key Store (JKS) used for holding trusted certificate keys. However, if you choose not to differentiate between personal keys and trusted keys, and opt to use a single key database for both tasks, this field should be set to the same value as the Key file name. In our example, this is set to `C:\WebSphere\AppServer\etc\WASLDAPKeyRing.jks`.

Trust file password: state the password used to protect the Java Key Store (JKS) certificate key database above.

Trust file format: ensure that **JKS** is selected.

Security level: setting this to High will ensure that the strongest SSL encryption algorithms are used for secure communication. The setting must be compatible with algorithms supported by the SSL peer.

3. Click **OK** when you are done.

> **Note:** If you are using the Default SSL settings for securing the LDAP connection, you have to restart the server before you can enable SSL for the LDAP User Registry for WebSphere Application Server V5.

LDAP User Registry

Here, the assumption is again made that you have previously configured WebSphere to successfully authenticate users against the SecureWay LDAP Directory Server without using SSL for securing the WebSphere-to-LDAP connection. The LDAP Distinguished Name (DN) and LDAP topology structure do not need to be modified in any way to support SSL.

Figure 10-59   LDAP User Registry Configuration

1. Launch the Administrative Console `http://<serverName>:9090/admin`. Click **Security -> User Registries -> LDAP**.

2. On the right hand frame, fill up the Configuration fields with as follows:

   Port: specify `636` which corresponds to the TCP/IP port listening for SSL enabled LDAP queries on the remote SecureWay LDAP Directory.

   SSL Enabled: select this check box to enable SSL.

   SSL Configuration: From the drop-down list you should see the LDAP SSL Entry we created previously, select it.

3. Click **Apply**.

4. Save the configuration.

5. Re-start WebSphere so that changes can be included next time.

Testing the connection

When the server starts, go to the Administrative Console, it should ask you for the user name and password for authentication. This is because Global Security is enabled. Give the user name and password as `wasadmin` (or `cn=wasadmin,o=ibm,c=us`) and password as `password`. If you are able to login successfully that means your configuration is working fine.

Disabling SecureWay Anonymous LDAP searches

In a production environment, it is envisaged that you may wish to prevent anonymous LDAP searches of the WebSphere user space, although such searches typically only reveal non-sensitive information about a user. The very act that any user information can be retrieved at all may pose a security risk.

The Netscape Address Book or Microsoft Outlook Address Book can be used to demonstrate this argument. With a little knowledge about the remote LDAP server, it is possible to retrieve the WebSphere authentication user registry.

10.14 JMX MBean security

Managed resources in WebSphere are represented by JMX MBeans. All these management interfaces in WebSphere are protected by security role based access control. All the MBean information is defined in the MBean XML descriptor file. This XML file is used to register these MBeans with WebSphere MBean Server. At runtime MBean descriptor information is processed and saved into ModelMBeanInfo instance.

WebSphere administrative subsystem supports four security roles: *Monitor role, Operator role, Configurator role and Administrator role.*

Table 10-8   Administrative roles

Role	Activity
Monitor role	View configuration information and status
Operator role	Trigger runtime state changes, such as start an application server or stop an application server
Configurator role	Modify configuration information but can not change runtime state
Administrator role	Operator as well as Configurator

Since most of the MBeans existing in WebSphere deal with runtime operations and runtime attributes and they only require monitor, operator or administrator role. Very few MBeans deal with configuration and would require monitor, administrator and configurator role.

Every method of MBean is categorized into one of the four types: INFO(read only like), ACTION(read and write), ACTION_INFO(read and write), and UNKNOWN(unknown nature).

Table 10-9   MBean method categories

Category	Operation
INFO	Read only
ACTION	read and write
ACTION_INFO	read and write
UNKNOWN	of unknown nature

MBean deployment descriptor XML file is very similar to EJB depoyment descriptor even though it does not support featues of EJB 2.0 such as unchecked method, excluded list , and run-as element. Run-as is not supported because administrative security is used to protect sensitive system resources and definitely it requires more privileges.

10.15 Cell Security

The Cell Security name refers to the scenario where security is enabled for a multi-server environment. In this environment WebSphere Application Servers are integrated into a Cell using the WebSphere Network Deployment (ND) package.
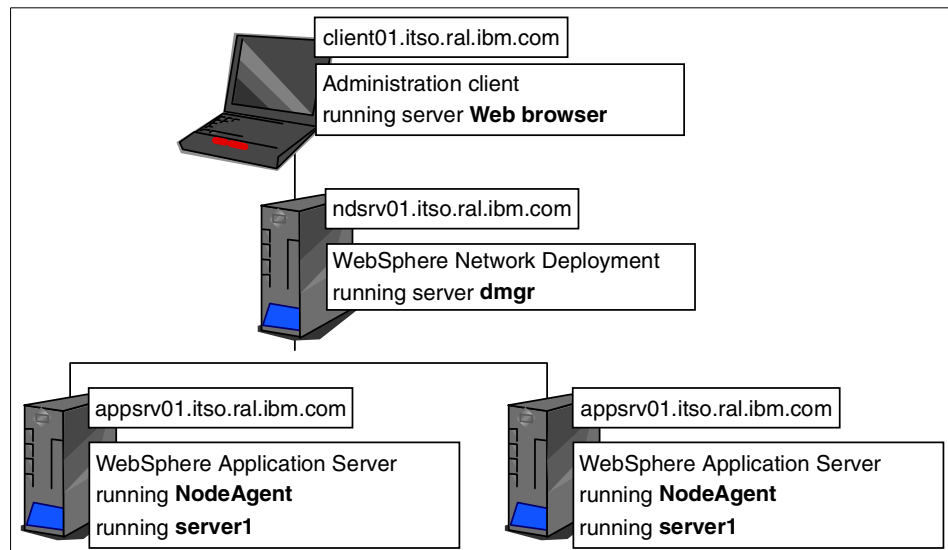
A sample scenario is depicted below.

Figure 10-60   WebSphere cell with one deployment manager and two application servers

There are differences in the runtime environment from the security and from the system management point of view, when you federate your application servers under one cell, managed by a Network Deployment manager.

► The administrator application disappears, gets uninstalled, from the individual nodes. All the management happens from the Network Deployment manager providing one single access point for administration.

► The embedded JMS servers are detached from the application servers. While the embedded JMS server depends on the application server when running a base application server; in a federated cell, the embedded JMS server is running separate from the application server, they are managed individually.

► In a federated cell only LTPA (Lightweight Third Party Authentication) is available as authentication mechanism for the individual servers. The main point is, that SWAM (Simple WebSphere Authentication Mechanism) cannot work anymore, because we need to pass credental information between servers.

   CSIv2 takes care of passing credentials between EJB containers.

   LTPA takes care of passing credentials between Web containers by enabling Single Sign-On for the Cell.

► Configuration files for the cell will be created and the configuration files for individual servers will change.

10.15.1 Configuring security for the Cell

There are three levels in the application server organization in WebSphere from the configuration point of view (starting from the lowest organizational level)

- ► Server, represents the application servers, JMS servers.

- ► Node, represents the node in the architecture, in teh network. From the administration point of view the Node is responsible to manage the multiple servers on the Node.

- ► Cell, represents multiple nodes collected into one centrally managed unit. From the administration point of view the Cell is responsible to manage the multiple nodes and servers in the Cell.

Security configuration, just like other configurations, can be defined on these levels. Although most of the settings apply to the server and cell level.

Administration

Once a node is attached to a cell, the servers are running within the cell, they can be only configured from the deployment manager. To configure any server in the cell go to the URL: https://<deployment_manager>:9090/admin, where <deployment_manager> is the host name of the deployment manager machine.

Security configurations in a cell

The following table is a collection of security items and settings in a WebSphere cell. You can read out from the table, where you can specify certain settings and what is the scope of these settings when administering a cell.

Table 10-10   Security settings in a cell

Item	Cell	Node	Server	Appl.
Enabling global security	X		X	
Enabling Java 2 security	X		X	
User registry	X			
Authentication Protocol	X			
CSIv2 protocol settings	X		X	
IBM SAS protocol settings	X		X	
JAAS Logon Modules configuration	X (C)*		X (A)**	
CosNaming Roles	X			
Administrative Role mappings	X			

Item	Cell	Node	Server	Appl.
User to role mapping				X
J2C Authentication Data entries	X (C)*		X (A)**	
SSL Repertoire entries	X (C)*		X (A)**	

(C)* means that the item was configured on a particular level and it is available on a different one.
(A)** means that the item is available on a particular level and it was configured on a different level.

> **Note:**
>
> J2C Authentication Data Entries are also used in the Resource Adapter Settings.
>
> The SSL Repertoire entries are also used in the Web container security, CSIv2 and IBM SAS transport protocol security.

Global security

The global security settings for a cell is almost the same as the global security settings for a server (not being in a cell). The difference is that only LTPA is available as authentication mechanism. The reason is that SWAM is not capable for passing credentials between multiple application servers, therefore we need an authentication mechanism that supports us with this feature, at this time LTPA is the only that gives us this function. As an alternative you can develop and use your own authentication mechanism, or in future releases Kerberos might provide this functionality.

For all the other global security settings refer to Section 10.2, "WebSphere Global Security" on page 235.

SSL settings

The SSL Repertoire entries are exactly the same as for a server (not being in a cell).

The entries defined for a cell are synchronized between the application servers attached to the cell, therefore the entries are available for the whole cell or for individual servers, see the individaul server settings later.

Configuring the SSL settings, adding a new entry on a cell level will be reflected in the server's security settings; but only the configuration is synchronized. It is the administrator's responsibility to make sure that the keys are copied to the right location for the application servers. Very important, that the SSL settings refer to certain directory paths, that are not the same on each server, simply because the WebSphere root direcotry is not or might not be the same on every machine, especially when heterogenous (UNIX, Windows) platforms are attached to the cell.

The solution for this problem is to use the WebSphere environment variables in path definitions on the server level and specify the platform and installation dependent directories there. Once an environment variable with the same name is defined for each server, you can refer to that variable on the cell level. This is what happens when you use to the ${WAS_ETC_DIR} variable for example.

For SSL settings refer to Section 10.8, "Configuring SSL" on page 257.

User registries
The user registry configured for the cell will be the user registry for each server in the cell.

The user registry for the cell should be a centralized repository, an LDAP Directory, OS users from a domain, or a custom user registry reading from a centralized, common, user repository.

For user registry settings refer to Section 10.4, "Configuring a user registry" on page 243.

Authentication Mechanisms
As mentioned before, in a cell LTPA is the only available authentication mechanism at this moment.

When you configure LTPA for the cell at the deployment manager, you will have to generate the LTPA key and secure it with a password. The LTPA private and public keys are stored in the security configuration, security.xml, file. Since the configurations in this file are synchronized, you do not have to worry about distributing the LTPA keys on each server, the deployment manager will take care of that.

For information on LTPA configuration refer to Section 10.6, "LTPA" on page 249.

JAAS configuration

The JAAS configuration settings, as Application Login Settings and J2C Authentication Data, are propagated to the servers and they are available on each end-point.

For more information about JAAS Configuration refer to Section 10.7, "JAAS Configuration" on page 254.

Authentication protocol

The authentication protocols configured for CSIv2 and IBM SAS:

► CSIv2 Inbound Authentication
► CSIv2 Outbound Authentication
► CSIv2 Inbound Transport
► CSIv2 Outbound Transport
► SAS Inbound
► SAS Outbound

These settings are also propagated to the servers, and they are available at the end-points.

For authentication protocol configuration refer to Section 10.12, "SSL between the Java client and WebSphere" on page 308.

After configuring security for the cell

Once security is enabled for the cell, you will have to restart all your components in the following order:

1. Stop the application servers.

2. Stop the node agents.

3. Restart the deployment manager.

4. Start the node agents.

5. Start the application servers.

10.15.2 Configuring security for an individual server

Servers in a cell can still have separate settings for security in a certain extent. Not all the security settings are available for a server when it is in a cell, some of the configurations are inherited, and can only be inherited from the cell. In order to configure security individually for a server, perform steps below.

1. Navigate to the **Servers -> Application Servers**.

2. Select the individual server you want to administer, for example: server1.

3. Click the link **Server Security** at the bottom of the page.

Figure 10-61   Server security settings in a cell

The following sections provide some more details on each settings available on the server level.

Use Cell Security
The Use Cell Security button will update the security settings for the server with the security settings from the cell; and remove all the individual settings.

Use Cell CSI
The Use Cell Security button will update the CSI settings for the server with the CSI settings from the cell; and remove all the individual settings.

Use Cell SAS
The Use Cell SAS button will update the IBM SAS settings for the server with the IBM SAS settings from the cell; and remove all the individual settings.

Individual CSI and SAS settings

The CSI and IBM SAS authentication and transport settings can be configured for each server individually. The following configurations can be individually set for a server.

- ► CSI Authentication -> Inbound
- ► CSI Authentication -> Outbound
- ► CSI Transport -> Inbound
- ► CSI Transport -> Outbound
- ► SAS Transport -> Inbound
- ► SAS Transport -> Outbound

These settings are the same as for a server (not in a cell). For more information about CSIv2 and IBM SAS settings, refer to Section 10.12, "SSL between the Java client and WebSphere" on page 308.

When configuring the transport for CSIv2 you can specify the SSL settings you want to use. The SSL settings available here are the list of SSL entries configured for the cell. For more information on the SSL settings for the cell, refer to Section , "SSL settings" on page 338.

Server level (global) security

Global security settings can also be configured for servers individually, although not all the configuration options are available on server level.

Select the **Server Level Security** from the Server Security page, you will get the panel as on Figure 10-62.

Figure 10-62   Server level security settings

The server security settings available here are the same as global security for a server (not in a cell).

The available security settings on the server security panel are:
- ► Enabled
- ► Enforce Java 2 security
- ► Use Domain Qualified User IDs
- ► Cache Timeout

- ▶ Issue Permission Warning
- ▶ Active Protocol

For more information about global security settings refer to Section 10.2, "WebSphere Global Security" on page 235.

Part 2

End-to-end security

345

11

Security in patterns for e-business

This chapter discusses the security considerations for end-to-end solutions in the context of the Patterns for e-business.

First a very short introduction to Patterns for e-business is given, describing what sort of patterns are used on what solution development stages. Then based on the ITSOBank sample application a high level solution design life cycle is presented based on selected patterns. All assumptions and decision drivers has been documented at each design stage.

At the end of the chapter final system architecture was proposed as a result of the process.

11.1 Patterns for e-business

Patterns for e-business is a group of reusable assets developed to facilitate and speed up the process of solution design for e-business application.

Patterns in a solution design process can be considered as an incomplete architecture. Depending on the pattern used, we can talk about logical (based on conceptual modules and building blocks) or physical / runtime architectures.

Difference between a pattern and an architecture is in the goal.

Pattern is usually used as a generic type of solution that can address a family of problems. It needs tailoring to make it effective solution and must be customized to include all the aspects that exist in the IT environment where it will be used.

Architecture describes the structure of a particular system that will be completed over time. Architecture is developed individually for each client as opposed to a pattern, which must be tailored to each client.

A number of different patterns has been developed and is still being extended for e-buiness applications. The layered model of patterns is depicted on the following diagram.
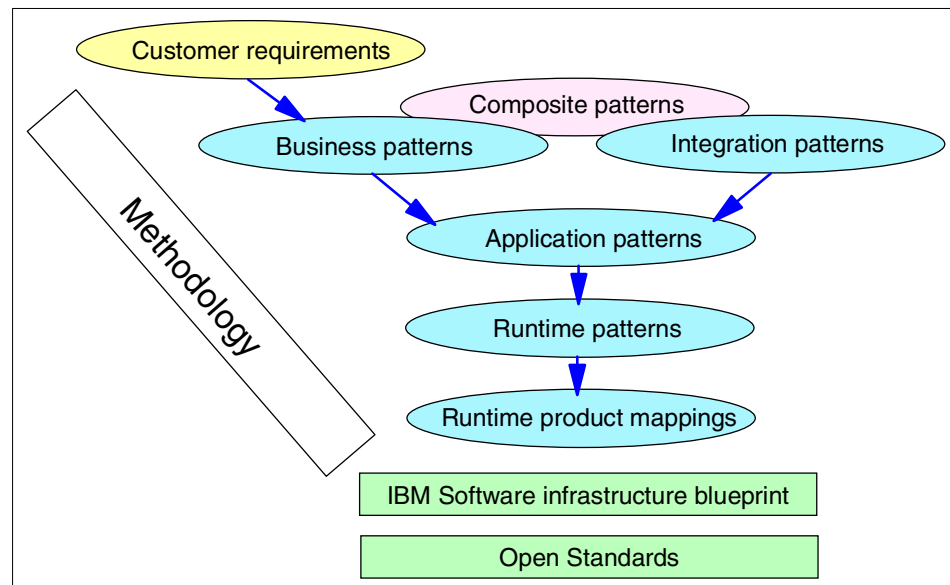
Figure 11-1   Layered model of Patterns for e-business

Business patterns

This is the view of the CIO on the business problem. It defines business processes and rules, actors of the system, user roles, existing and future support modules as "architectural" building blocks.

In this category we may identify following patterns:

- ▶ Self-Service
- ▶ Collaboration pattern
- ▶ Information aggregation
- ▶ Extended enterprise

Integration Patterns

This view defines how the business actors interact with business processes and supporting tools. It describes what kind of integration is required in front-end and back-end applications combining in this way the business patterns to form a solid solution. Integration patterns are often used as a "glue" between different business patterns. They are not directly related to a specific business processes.

In this category we may identify the following types of patterns:

- ▶ Access Integration: Single Sign-On, personalization, content adaptation and so on.
- ▶ Application Integration: all kinds of interaction between applications and applications and data.

Composite Patterns

Composite Patterns are a combination of Business and Integration Patterns. This combination sometimes forms a advanced e-business application and therefore it has been introduced as a separate pattern that can be used as a starting point for design process.

Among different composite patterns we may identify:

- ▶ Corporate portal patterns
- ▶ E-commerce patterns
- ▶ Application Service Provider patterns

Application and Runtime Patterns

Application and Runtime patterns are driven by the customer's requirements, they describe the shape of applications and the supporting runtime needed to build the e-business solution. In fact they define how to implement the combined Business and Integration patterns.

Application patterns are chosen after selecting business, integration or composite patterns and they lead to runtime patterns. Runtime patterns are introduced to group functional groups into physical nodes. The nodes are interconnected to solve the initial business problem.

For more information on patterns refer to other documents at Section 11.6, "More information on patterns for e-business" on page 365.

> **Note:** The ITSOBank sample application presented in this book is a simple example of a J2EE application, focusing mostly on the technology and security features, used to access customer account and transfer funds. We will use this example throughout the chapter to present patterns related aspects in the design process. Although it can not be considered as real life solution reference and should not be used as a reference. It does not take into account many customers related aspects and custom requirements that may significantly influence the final design approach.

11.1.1 Patterns and solution design process

The full solution life cycle process should include the review and selection of each Business, Integration and Composite patterns and adaptation to the customer needs. The process can be often shortened, when we already know many details about the solution. In such a case we can jump directly to a specific design level.

As the main subject of this book is security, we will not go through the full life cycle process starting from business requirements and integration aspects. We will skip these subjects and will go directly to the selection of appropriate business and integration patterns. However few assumptions need to be made to consolidate our understanding and lead our future decisions.

Basic business drivers

A few business driver that has been addressed on the implementation level in the ITSOBank sample applicaiton, provided with this book.

► Application end users need to interact directly with business process provided by the application.

► There are multiple applications taking part in the solution; these applications need to be integrated.

► Application interface is Web based.

► Single delivery channel is provided for the Web application that enables business process.

► Business process latency must be reduced to minimum, back-end integration will be necessary to support Web enabled business processes.

IT drivers
A few IT drivers that has been addressed on the implementation level in the ITSOBank sample applicaiton, provided wit this book.

► System provides Single Sign-On across different applications.

► Application is structured in layers, with separate presentation logic, business logic, and data and enterprise integration logic.

► Users directory is centralized and accessible for all applications.

► Security context is propagated across entire business process.

► Administration and maintenance costs should be as little as possible.

Based on these drivers first decision that we made is that the base for the sample application isa combination of the Self Service Business Pattern and Extended Single Sign-On Integration Pattern.

11.2 Selecting application patterns for ITSOBank

Application patterns show principal layout of the application, focusing on the shape of the application, the application logic and associated data. This section summarizes decision made for selecting application patterns from business point of view and IT / technology point of view.

Application pattern for Self Service Business pattern
A decision was made to use Directly Integrated Single Channel application pattern for the ITSOBank sample application.
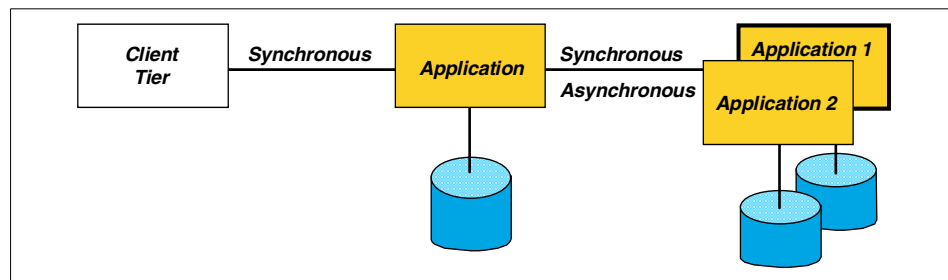
Figure 11-2   Directly Integrated Single Channel Application Pattern

The primary business driver that directed the use of this pattern is to reduce business process latency by providing direct access to back-end application and data from the Web enabled business process.

Besides, this pattern provides a structure for point to point connection from client browser to one or more applications.

We do not need multiple channel integration mechanism. Although this application pattern can be used to implement any one of the delivery channels we only focus our design on Web delivery channel.

This application pattern requires applications to be divided into three different logical tiers: presentation, Web application and back-end business logic. Data can be accessed from both Web and business tiers. We can think of any typical J2EE structure while selecting this pattern.

► Presentation tier is responsible for all the application presentation logic.

► In Web application tier some of the application business logic is done and all the communication to the back-end business logic is implemented.

► Back-end application logic in our case covers data access functions of the ITSOBank application. Here are all the application integration interfaces implemented as well.

All the communication between tiers, as it was presented in the picture is either synchronous or asynchronous.

Application pattern for Extended Single Sign-On Integration Pattern

Main goal of the Access Integration Pattern is to give users consistent and seamless front-end access mechanism to multiple business applications that reside on multiple servers, each with its own access mechanism.

In our ITSOBank example we have chosen the Extended Single Sign-On application pattern.
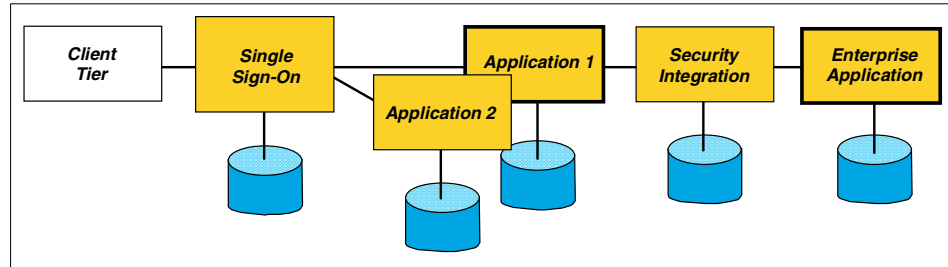
Figure 11-3   Extended Single Sign-On application pattern

The Extended Single Sign-On Pattern enhances the Web Single Sign-On Pattern with propagation of the security context. In a Web Single Sign-On pattern sign-on functions are performed in Web tier. In Extended Single Sign-On, as shown in the picture above sign-on functions are externalized and almost always are based on central user registry. This introduces more flexibility in fulfilling privacy and security auditing requirements.

Main drivers that directed our decisions were:

► Users will have seamless access to WebSphere and Domino based applications and data without being prompted for login name and password by each application separately.

► External authentication and authorization system will authenticate the user and set up security context that can be propagated through the entire business process from the Web tier down to the business logic and back-end tier.

► Reduced maintenance and administration costs for authentication and authorization system thanks to the centralization of the user registry and security policy database.

► Reduced total cost of ownership thanks to the simplification and improved efficiency of the users' and security policy management.

The following tiers can be distinguish in this pattern:

► Client tier - similarly to Self-Service Application Pattern this tier represents end user interface used to interact with the application. In the case of ITSO it will be Web browser interface.

► Single Sign-on tier, which from the security point of view is the main component of the solution design. It is responsible for authenticating users and establishing security credentials, as well as ensuring seamless sign-on capability across multiple applications. In this tier security administration and policies are implemented. Sign-On tier uses centralized users' and policy database. In the case of ITSOBank it is a LDAP server user registry.

► Application tier may represent a new or existing application that will be part of the Single Sign-On domain. In our case these applications are WebSphere J2EE ITSOBank application and a simple domino application used with ITSOBank.

11.3 Creating common runtime pattern for the ITSOBank application

As mentioned in previous sections runtime patterns define functional nodes that underpin an Application pattern. Runtime patterns represent very high level physical solution architecture, where networks and nodes are identified but no product selection has been done. Most runtime patterns will consist of the core set of nodes used for Web application design with additional nodes specific to the customer situation.

In this section we briefly describe runtime pattern for both Self-Service and Extended Single Sign-On application. At the end of this section we will provide a possible runtime pattern for the ITSOBank sample application. This combined runtime pattern will be used in the next section also to map certain products and to describe the security flow between the nodes.

Runtime pattern for Self Service application pattern

Our runtime pattern for Self Service application is based on a simple three tier architecture where the Web server is separated from Web application server. The Web server resides in the demilitarized zone and is responsible for the Web presentation logic. Application server runs application business logic and communicates with the back-end systems and databases.

The LDAP directory is installed behind the domain firewall and is used to store user information related to authentication and authorization.
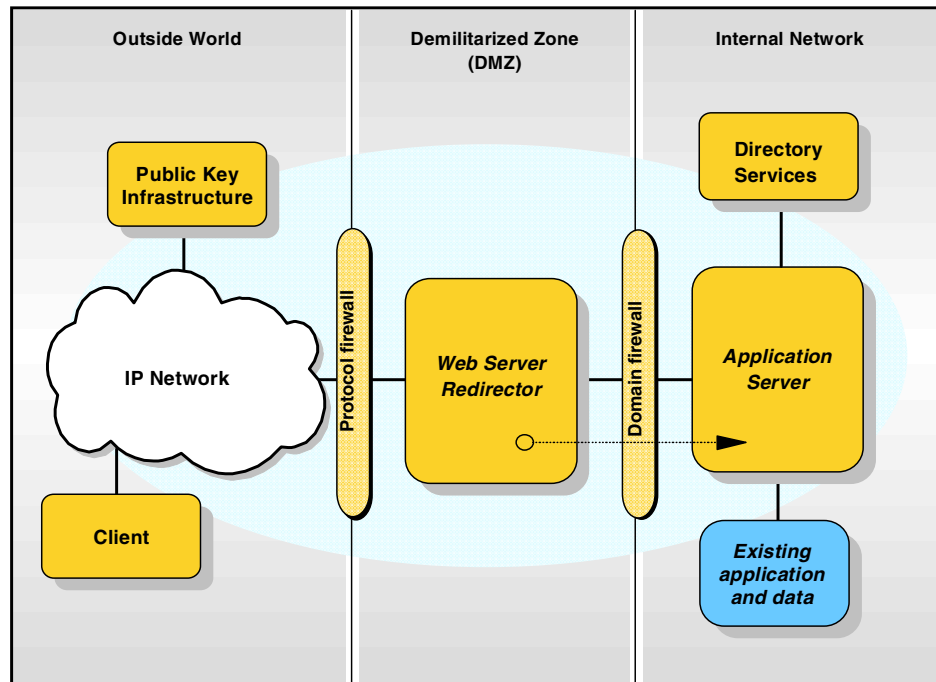
Figure 11-4   Runtime pattern for s Self-Service application pattern

The main nodes of the pattern are listed below, and described from a security point of view.

► The protocol firewall prevents unauthorized access from the Internet to the demilitarized zone. The role of this node is to allow the internet traffic access only on certain ports and block other ports.

► Web Server Redirector; in order to separate Web server from application server, Web server redirector has been introduced. Its job is to redirect the application requests to the application server node. Advantage of using Web server redirector is that we can move application server and all the application business logic behind the domain firewall

► The domain firewall prevents unauthorized access from the demilitarized zone to internal network. The role of this firewall is to allow the network traffic originated only from the demilitarized zone and not from the Internet.

► Application server provides the infrastructure to run application logic and communicate with internal back-end systems and databases.

► Directory services provides the information about the users and theirs rights for the Web application. The information may contain user's IDs passwords,

certificates, access groups etc. This node supplies the information to the security services the authentication and authorization service.

► The existing application and data node depicts the back-end systems and databases that are accessible from the Web application.

For more detailed description of run time pattern variation for Self Service application pattern please refer to 11.6, "More information on patterns for e-business" on page 365.

Runtime pattern for Extended Single Sign-On application pattern

The key consideration in choosing runtime pattern for Single Sign-On solution is diversity of e-business environment, types of application servers used and security management.

We have made a decision to use two different application servers for different functions of the application. This implies that we need to have an external security server, that will serve as a security proxy, which intercepts a request in order to map/transform user's credentials into the appropriate credential format acceptable by application servers. To support this part of our design we can use runtime pattern for heterogeneous servers with external authentication and authorization servers presented on the picture below.
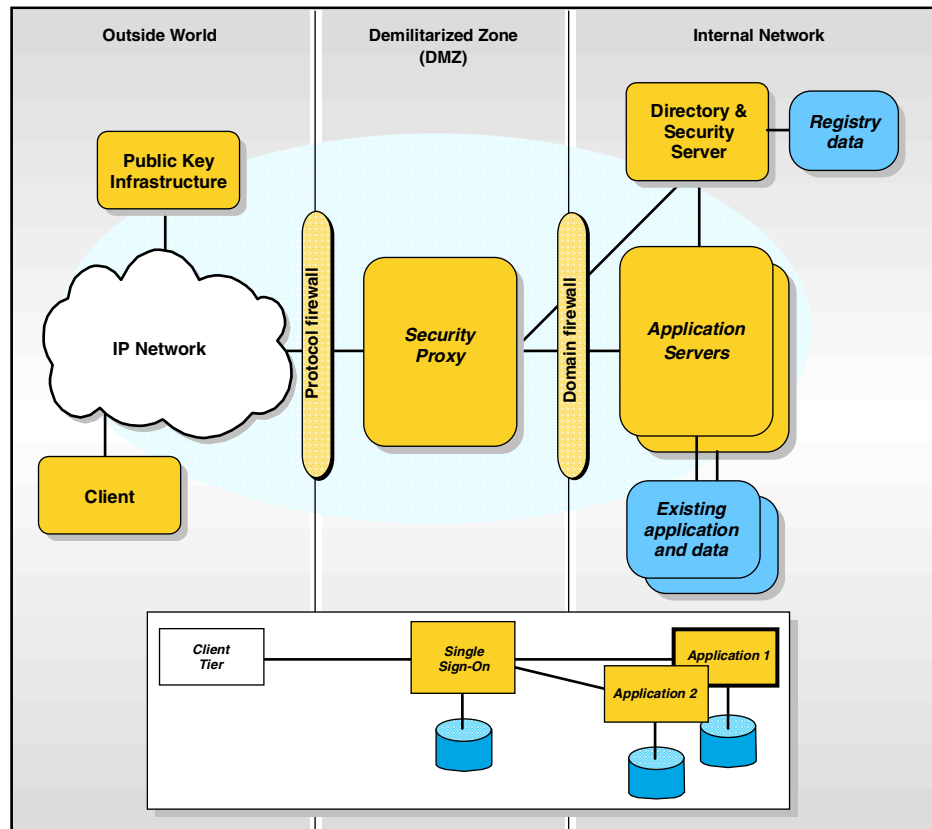
Figure 11-5   Single Sign-On runtime pattern for heterogeneous application servers

Another decision that was made was credential propagation from the Web-tier down to business logic tier and back-end applications. This will enable non-repudiation of back-end transactions initiated by Web users. We can achieve this by using the same security server to manage the Web tier and business logic and back-end applications. No credential mapping or transformation is required; this might be an option if separate security mechanisms would be used in different application tiers. Security context is presented all the way from the Web down to the back-end resources.

This approach significantly simplifies user's and policy management by unifying user profile across entire business process supported by Web application. It however requires sometime complex configurations and usage of security servers that are supported by the chosen application servers.

Diagram below presents runtime pattern for extended Single Sign-On solution where heterogeneous application servers are used and an external security server provides security management for all application tiers.
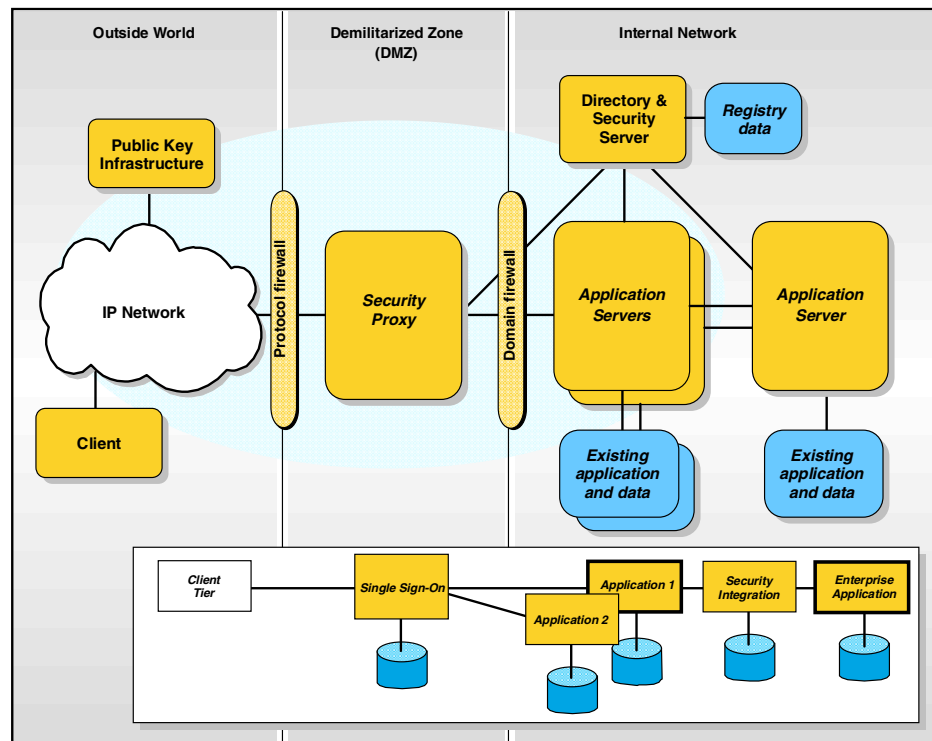
Figure 11-6   Extended Single Sign-On runtime pattern for central security service

Nodes used in this last runtime pattern are:

► Security Proxy; the role of the security proxy is to intercept incoming request and map or transform user credentials into the format acceptable by the application server that was the original target of the request. The security proxy is used to implement Single Sign-On between heterogeneous Web application servers.

► Application Server - this node includes the application server that runs the application logic for the solution. It can be installed in the same machine with the Web server or separated by a domain firewall as described in the Self-Service runtime pattern section.

► The Directory and Security Server provides the information about the users and theirs rights for the application. The information may contain user's IDs passwords, certificates, access groups and so on. This node supplies the

information to the security services like authentication and authorization service.

Combined runtime pattern for the ITSOBank sample application

The ITSOBank sample application has only a few function, it was designed to show the security capabilities of a J2EE application, WebSphere Application Server V5 and some of the end-to-end security design considerations.

Please note that functionality of the application as implemented in this book does not fully reflect the real world scenarios. Therefore this section should be rather used as a reference to implement similar solutions.
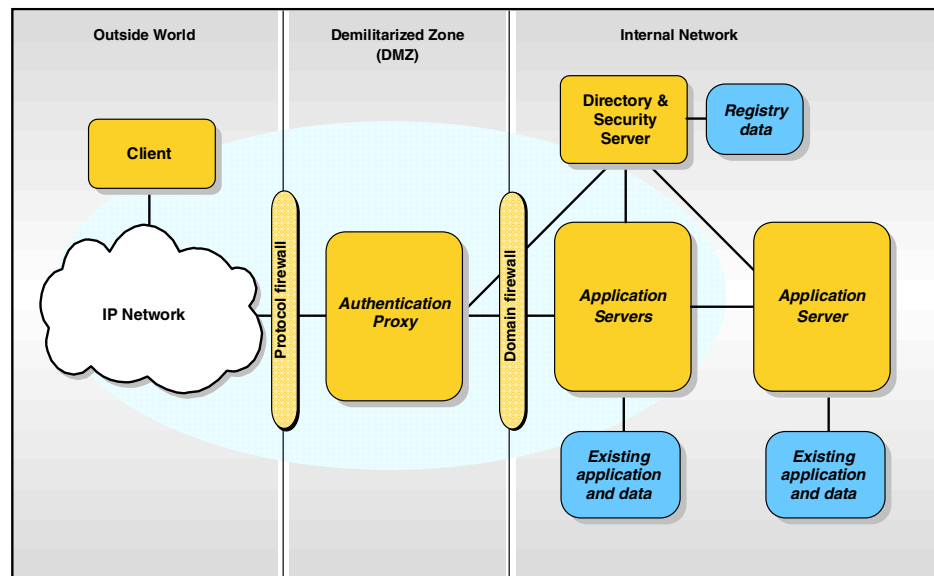
Figure 11-7   Runtime pattern for ITSOBank application

The sample application fit into the runtime application pattern from Section , "Runtime pattern for Extended Single Sign-On application pattern" on page 356.

11.4 Product mappings

This section presents an example of product mappings for the runtime pattern that we have shown in the previous sections. We will focus only on the customer based flow. We will not cover other flows that should be considered when working on real life solution, like employees flow and partners flow.

Product mappings for the ITSOBank sample application

Product mappings used in the scenario is presented in the diagram below.
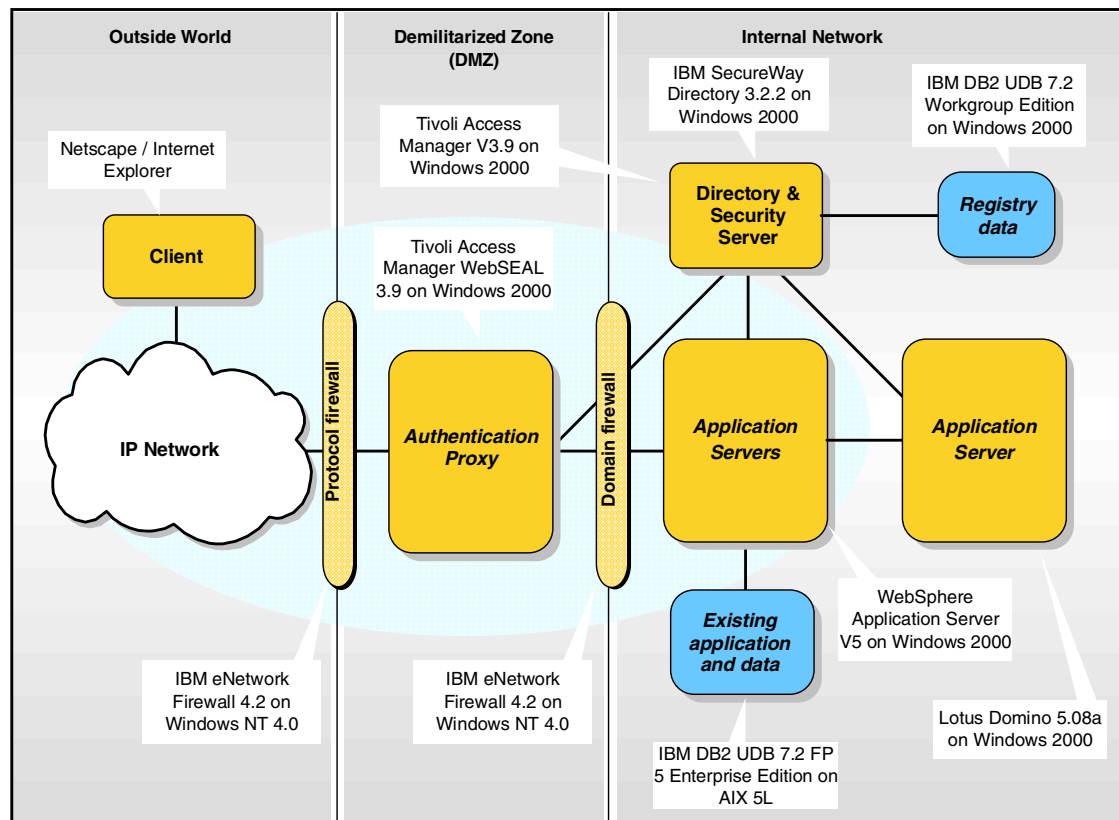
Figure 11-8   Product mappings for ITSOBank application scenario

The products depicted on the picture above are:

▶   Tivoli Access Manager's WebSEAL

WebSeal is a security reverse proxy that is used to authenticate the user, create and maintain a session with the user and provide URL level authorization. It also hides the internal structure of Web resources through URL mapping. WebSeal supports multiple types of authentication and implements stepping up to stronger authentication type if necessary.

► IBM HTTP Server

The Tivoli Access Manager can protect any static content on the Web server including the server itself, so that non authenticated users will not be able to communicate with the Web server behind the security reverse proxy, WebSEAL.

► WebSphere Application Server V5

This server runs a main application logic of the ITSOBank sample application. For passing credentials between multiple application servers, other WebSphere servers and Domino, WebSphere is using LTPA tokens.

► Lotus Domino Application Server

Domino Server runs a component of the ITSOBank sample application. WebSphere and Domino estabilish the Single Sign-On using LTPA tokens.

► SecureWay Directory Server

Access Manager supports a number of LDAP directories. The IBM SecureWay LDAP Directory is shipped with the Tivoli Access Manager; it stores user information and user privileges; besides other application information.

► IBM DB2

Internal systems are represented in our scenario by application database that is stored on DB2 server.

► Tivoli Access Manager

Tivoli Access Manager consists of the following runtime components:

– Management Server

The Management Server is used to manage the Access Manager security policy. The Management Server receives updates from the console, Administration API or Administration command line interface.

– Authorization Server

The Authorization Servers are used by applications in remote mode. Remote mode means that the application sends a request to the server to answer the question "Can the user perform the action on the resources". Local mode means that the application has an in-memory cache of the policy so the application can check this for a decision without sending a message outside the application; for example: WebSEAL works in local mode.

For more information on the Tivoli Access Manager product and integration with WebSphere Application server please refer to the Chapter 12, "Tivoli Access Manager" on page 367.

The next steps provide a simple technical walkthrough for user authentication in the sample application, that you can also follow on Figure 11-8 on page 360.

1. Customer (application end user) will be using a browser to locate the Web application from the Web.

2. The request hits the protocol firewall that only allows appropriate traffic. From here, the traffic is passed to security reverse proxy. An extension to this can be to implement network dispatcher that would select the Web reverse proxy that is most available at a time.

3. The security reverse proxy is responsible for authentication and for session establishment and maintenance. The proxy will authenticate the user if it is required for the resource, then establishes the session. Authentication is checked against the LDAP user registry.

4. Once a session is established the security reverse proxy will authorize the user based on the URL the user is trying to access. This authorization is coarse grained as it can only affect the URL requested.

5. If the request is authorized then it is forwarded to the Web server. The reverse proxy may perform load balance across the Web servers. An extension can be introduced here for managing the load between Web servers by introducing load balancer between the security reverse proxy and the Web servers.

6. The request is then sent through the second firewall to the application servers. The Web application servers execute business logic and call on the authorization service for finer grained control. This authorization service will be accessible via an API or through standard J2EE security. If the request is authorized then a function will be executed on behalf of the authenticated user. If the function communicates with back-end system through integration server, then it is up to the design of the integration layer to call authorization service for further, finer, levels of authorization.

11.5 Security guidelines in Patterns for e-business

The Patters for e-business design approach also provides guidelines for solution design. The guidelines consist of technology options, application design and development, security, performance and availability, system management. The following short sections will point out some common security guidelines that should be taken into considertion while designing an e-business solution.

Securing connections in a solution

On the architecture level, as opposed to the application level, connections between nodes should be secured. For information in relation with this topic; on the application level, where secure connections should be ensured between modules; refer to the Section 3.1, "J2EE Application" on page 22.

The purpose of securing the communication is to prevent non authorized persons and systems to listen the communication or to participate in the interaction.

The following diagram depicts the commonly used and highly recommended secure communication lines between nodes.
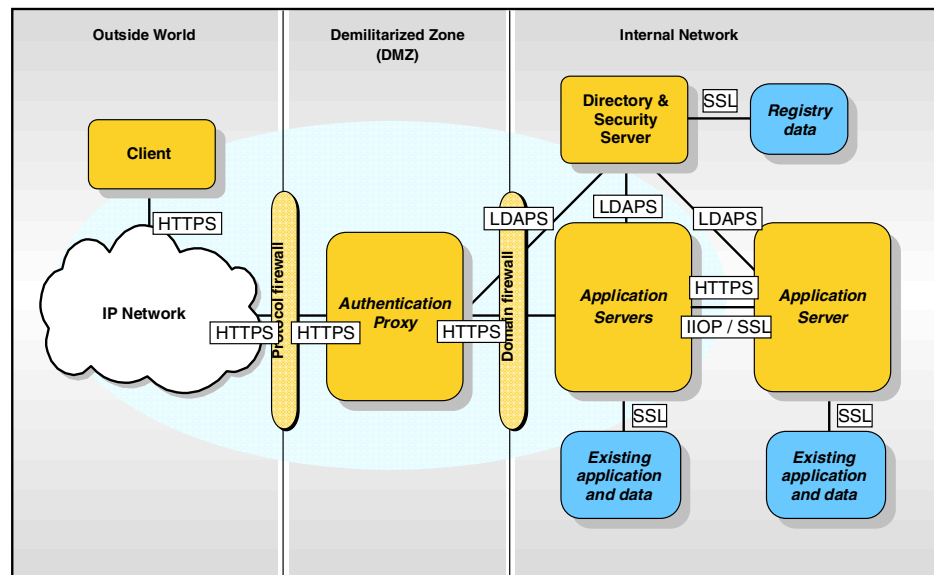

Figure 11-9   Secure connection between nodes

The following secure communications are identified on the diagram above:

▶ HTTPS is the secure HTTP connection using SSL. Nodes, that are communication via TCP/IP using HTTP protocol, should use secure SSL communication. The level of security depends on the options set for the connection.

▶ LDAPS is the secure LDAP connection to a directory server using SSL. Since LDAP direcotries store essential and sensitive applications, business information; the communication should be secured.

▶ IIOP/SSL (IIOPS) is the secure communication for IIOP connections using SSL. Two application servers are mostly communicating via IIOP, for example EJB client and EJB container.

> **Note:** Two application servers can also communication via HTTP with SOAP using the Web Services technology. The HTTP communication should be secured using SSL.

▶ SSL is a transport layer security protocol, that can be applied to most of the protocols in use with an e-business application. On the diagram above other connections without named protocols can use also SSL to secure the communication.

Other communication channels between nodes can be secured on a transport layer, for example using IPSEC.

System hardening

Beside securing the communication that secures the nodes from being attacked from outside; systems have to be secured from inside attacks as well. Operating Systems security is an essential part of every system and provided as mandatory. System hardening is a global philosophy of system security that focuses strongly not only on detection, but also on prevention. It involves removing unnecessary services from the base operating system, restricting user access to the system, enforcing password restrictions, controlling user and group rights, and enabling system accounting.

System administrators are responsibile to follow the system and corporate guidelines to ensure security on every level. System security has to be maintained and set correctly. Part of system security is to harden the system and prevent attacks from both inside and outside.

System hardening relies on the system management guidelines and the advanced security settings and functions provided by the system.

Applications have to be in sync with the system security. Although sometimes the applications require some flexibility from the security side; the reason can be unresolved design issue or special requirements. These cases can open security holes or can weaken the system security if they are not monitored not maintained correctly.

11.6 More information on patterns for e-business

The most valuable source of information about patterns is the book *Patterns for e-business: A Strategy for Reuse* by Jonathan Adams, George Galambos, Srinivas Koushik, and Guru Vasudeva, ISBN 1931182027.

For more information about Access Integration security, refer to the following IBM Redbook *Applying the Patterns for e-business to Domino and WebSphere Scenarios*, SG24-6255.

For more information about Patterns for e-business, see the following Web site: http://www-106.ibm.com/developerworks/patterns, or refer to the following IBM Redbooks:

- *Applying the Patterns for e-business to Domino and WebSphere Scenarios*, SG24-6255

- *Self-Service Patterns using WebSphere Application Server V4.0*, SG24-6175

- *User-to-Business Pattern Using WebSphere Personalization Patterns for e-business Series*, SG24-6213

- *Self-Service applications using IBM WebSphere V4.0 and IBM MQSeries Integrator Patterns for e-business Series*, SG24-6160

- *e-commerce Patterns for Building B2C Web Sites, Using IBM WebSphere Commerce Suite V5.1*, SG24-6180

- *Access Integration Pattern using IBM WebSphere Portal Server*, SG24-6267

- *Mobile Applications with IBM WebSphere Everyplace Access Design and Development*, SG24-6259

12

Tivoli Access Manager

This chapter will attempt to address some concerns of end-to-end security, not just for a single WebSphere-hosted application but also for an enterprise's total e-business offering.

► Security considerations around the infrastrural design supporting an application, including operational and personnel issues.

► Architectural considerations around enterprise user registries and the externalization of security services and policies.

Tivoli Access Manager can address both sets of considerations. As a security management application, it offers centralized authentication and authorization for WebSphere-hosted applications which may be leveraged to provide at least a stepping stone on the path to the Holy Grail of Enterprise Single Sign-On.

We will also look some possibilities for integrating WebSphere Application Server and Access Manger infrastructural components in order to provide a common security policy across an enterprise.

12.1 End to End Security

This part of the book largely concentrates on securing WebSphere hosted applications, however the application is only one part of the e-business infrastructure which provides the services to support the publication of applications to their intended user audience.

In order to provide a secure solution the entire infrastructure and the flow of data through the infrastructure must be examined for possible breeches. Best practices require that a complete Risk Analysis be carried out and Risk Mitigation processes implemented with the remaining risks proactively monitored and the entire system regularly audited.

Security needs to be addressed at several levels, physical access, network access, platform operating system(s), application services throughout the infrastructure, for example: Web server software, middleware connectors and messaging infrastructure, and trusted operational personnel.

Each level must be addressed both independently of and together with the others. The most trivial example being a system where all other levels had been addressed except the personnel.

Corrupt and/or malicious employees with authorized access to a system are the single greatest security threat and apart from proactive auditing there is little that can be effective from a technological solution.

Often overlooked, access to the physical elements of a system can open a system to attack both by intruders, people who should not have physical access and in the more common case by otherwise authorized personnel. Once direct access to either the servers or the network devices, for example the hub/switch to which clustered application servers are connected, is obtained then all the other methods of attack become so much easier.

If the platform operating systems have not been "hardened" then free range administrative and diagnostic tools installed by default can be used both to cause damage and compromise information, either by changing or stealing it. "Hardening" systems at this level must include file permissions and passwords. Particular care must be taken with "remote" administration tools, be they browser or thick client accessed.

The network level is popularly thought to be most often attacked, at least as represented in the popular media. After all the point of e-business applications is to publish them so that Internet access for intended audience is available. Attacks such as Denial of Service (DoS), where the server is relentleSSLy bombarded with thousands of spurious requests with the intention of flooding the

bandwidth to and from the server until it can no longer handle legitimate requests, are not strictly in the baliwick of security as no information is changed or stolen. Real time operational performance reporting and pre-arranged re-routing procedures are the only real defense. IP spoofing, impersonating another legitimate connection and thereby by passing the IP and protocol filters of firewalls is of more concern and so IP address alone cannot be reliably used to identify authorized connections unless you are absolutely certain you have a trusted path between both the source machine and the target.

Where physical access is compromised, network sniffing, inserting a device or software which reads all the traffic between two points for later analysis or even real time substitution becomes possible. Virtual Private Networks (VPN) and cryptographic technologies can address some of these issues on a point to point basis between pieces of the infrastructure. A "trusted path", i.e. an un-encrypted or non-certificate passed connection can only be considered within Region 2 DMZ's when physical access and personnel are absolutely controlled, for example: a crossover cable from a firewall port to a security proxy.

Exploitation of bugs in the software services throughout the infrastructure are very real. In order to secure systems from this kind of attack, operational procedures to keep patches and fixes up to date must be in place and the implementation of a "defense in depth" architecture such as illustrated in Figure 12-1 on page 372, where there are multiple physical and logical layers that an attack must compromise each in turn before gaining access to the application and its information.

Each segment of the infrastructure which supports an e-business application must be analyzed for possible risk and the overall design of your system include risk mitigation methods at each point.

12.2 Network Identity and Centralized Security Services

Tivoli Access Manager for e-business V3.9 is the current name for what in immediately previous versions has been Tivoli SecureWay Policy Director. 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.

The rename is significant as it highlights one of the overriding concerns for any enterprise with multiple Web based applications - how do you control access across your entire e-business infrastructure without multiple and possibly conflicting security policies?

There is a business wide change in focus from implementing application specific security in order to prevent inappropriate users from accessing resources towards attempting to develop both 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 once, i.e. 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 given dependent on the security policy for those resources, i.e. Authorization.

The externalized security provided by Access Manager includes strategies to include legacy applications in Single Sign-On solutions through integration with pre-existing user registries and authorization databases.

If regardless of which application a user accesses within an enterprise, they always log on with the same id and password, (albeit there may be the requirement for stronger or re-authentication, perhaps token or certificate based around particularly sensitive information or high value transactions) then that consistent user experience appears from the user's viewpoint at least, as Single Sign-On. 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.

► 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 as 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 as well as a single point to apply policy.

► All of these benefits contribute to enabling an enterprise to be faster to market with new applications and features.

A down side here, of having implemented a single security solution which is based on a single technology or product, is that any product specific security exploitation results in an enterprise wide vulnerability. It does however let you concentrate your defences rather than being forced to dissipate your efforts across multiple platforms and products.

12.3 Tivoli Access Manager

While this book is not the place for a full blown "how to" install a Tivoli Access Manager Security Domain, the strength and flexibility of the solution will be indicated by the breadth of available components in the Tivoli suite and an enterprise strength design idealized for the leverage of WebSphere-hosted applications within a Single Sign-On environment will be described.

> **Note:** For installation instructions see the original product documentation that comes with the package or read the documentation at http://www.tivoli.com/support/public/Prodman/public_manuals/td/AccessMana gerfore-business3.9.html.

The Access Manager Secure Domain provides a secure computing environment in which Access Manager enforces security policies for authentication, authorization, and access control. Ignoring performance, redundancy and availability considerations which must be addressed in production systems, the essential components can be see in Figure 12-1 on page 372.
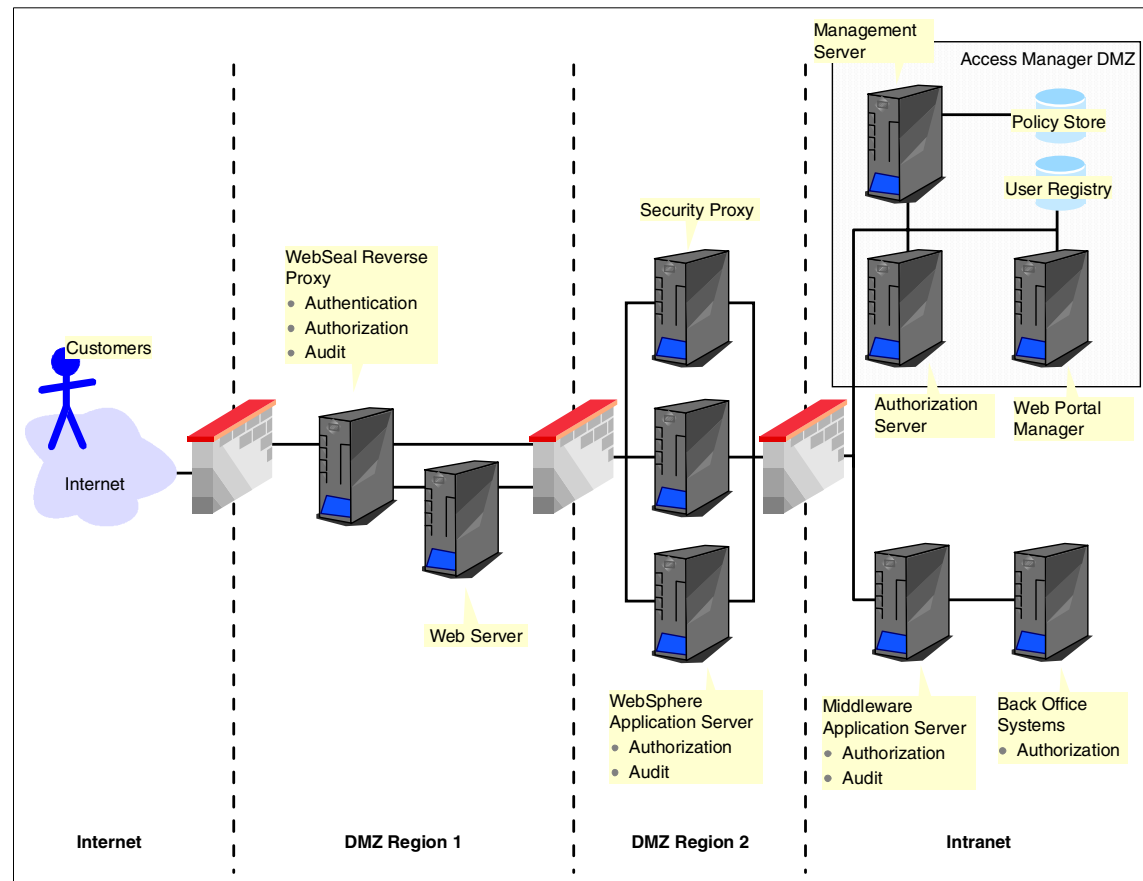
Figure 12-1   Typical Three-Tier infrastructure supporting e-business applications

Access Manager requires a User Registry and can be configured to use a variety, including Microsoft Active Directory and iPlanet but ships with IBM SecureWay LDAP Directory V 3.2.2. underpinned by 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 credentials information of all participants within the secure domain, both users and servers. A secure domain contains physical resources requiring protection. These resources include programs, files and directories. A virtual representation of these resources, protected by attaching ACL and POP policies is stored by the Policy Server.

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 authorization APIs which can be used programmatically within other applications and clients. Client calls for authorization decisions, through the Access Manager Run-time service, which must be on every server participating in the secure domain, are always referred to an Authorization Server. Programatically made calls can be local or remote Authorization Server. When running local node API, the application communicates to the security server (Access Manager), no authorization sever is required.

Authorization servers are the decision-making servers that determines 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 as well as the user registry. There is also a command line utility, pdadmin, which extends the available commands available to include the creation and registration of authentication blades such as WebSEAL which will be described a little later.

Access Manager can be configured 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

► BEA Robotic Application Server

► Web Server Plug-in which supports IIS 5.0 for a Windows 2000 Server/Advanced Server environment, iPlanet 6.0 for Solaris Operating Environment 7 (sparc) and IHS 1.3.19 for an AIX 5L environment.

The list of point products and components that are shipped in the Tivoli Access Manager V3.9 package can be found in the following table Table 12-1 on page 374.

Table 12-1   Access Manager Components as shipped with Version

Server	Required Component
IBM Secureway LDAP Directory V3.2.2	SecureWay Directory Server
	SecureWay Directory Client
	DB2 Universal Database Edition
	Global Security Toolkit
	HTTP Server
Tivoli Access Manager Policy Server V3.9	Access Manager Runtime
	Access Manager policy Server
	Global Security Toolkit
	SecureWay Directory Client
Tivoli Access Manager Authorization Server	Access Manager Authorization Server
	Access Manager Runtime
	Global Security Toolkit
	SecureWay Directory Client
Tivoli Access Manager Runtime Server (The minimum install on an application server to utilize the programmatic API's)	Access Manager Runtime
	Global Security Toolkit
	SecureWay Directory Client
Tivoli Access Manager Web Portal Manager	Web Portal Manager (WebSphere enterprise application)
	WebSphere Application Server
	Global Security Toolkit
	SecureWay Directory Client
	Access Manager Runtime
Tivoli Access Manager Application Development Kit (This is for a development environment may also include an installation of TAM's)	Access Manager ADK
	Access Manager Runtime
	Global Security Toolkit
Web Portal Manager and/or WebSEAL	SecureWay Directory Client

Server	Required Component
Tivoli Access Manager WebSEAL Server	Access Manager WebSEAL Server
	Access Manager Runtime
	Global Security Toolkit
	SecureWay Directory Client
	Access Manager Java Runtime Environment (If you have developed a CDAS using the Access Manager JRE then it is required)
Tivoli Access Manager for WebSphere Application Server (These 4 components are optional and together build an Access Manager Authorization Server which will optimize performance if installed on the same host)	WebSphere Application Server V5
	Access Manager Java Runtime Environment
	Access Manager for WebSphere Module
	Access Manager Authorization Server
	Access Manager Runtime
	Global Security Toolkit
	SecureWay Directory Client

These were the installed components for an Access Manager Secure Domain built on a platform running Microsoft Windows 2000 Advanced Server as shown in Figure 12-2 on page 376, with the exception of an AIX BD2 Database server for the ITSOBank sample application.

12.3.1 Environment for the scenarios

The following Figure 12-2 shows the lab setup for this particular project. This diagram is provided as a reference for later sections, discussing the different Tivoli Access Manager scenarios. It also gives a better understanding of the project architecture.
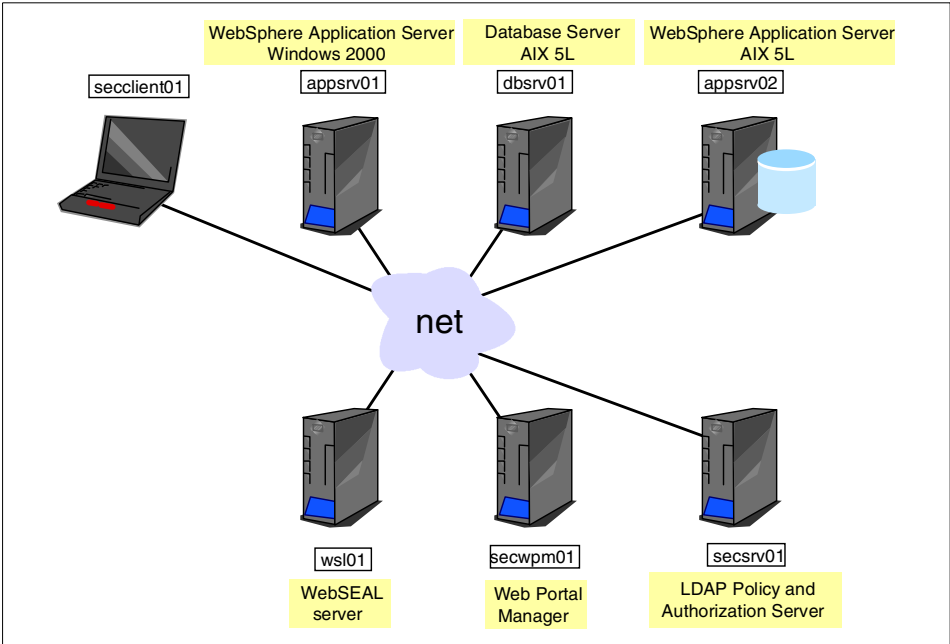
Figure 12-2   Lab Setup for the redbook project

The following table shows the different servers with the applications listed running on each machine.

Table 12-2   Application Servers as shown in Figure 2

Sever Name	Application Servers Running
secsrv01	IBM Secureway LDAP Directory V3.2.2
	Tivoli Access Manager Policy Server V3.9
	Tivoli Access Manager Authorization Server V3.9
wsl01	Tivoli Access Manager WebSEAL Server V3.9
	Tivoli Access Manager Application Development Kit V3.9
secwpm01	Tivoli Access Manager Web Portal Manager V3.9

Sever Name	Application Servers Running
appsrv01, appsrv02	WebSphere Application Server V5
	IBM HTTP Server
	Tivoli Access Manager for WebSphere Application Server V3.9
	Tivoli Access Manager Authorization Server V3.9
dbsrv01	DB2 Server for ITSOBank

12.4 Scenarios

This chapter covers four different scenarios with Tivoli Access Manager related to WebSphere Application Server integration.

► Scenario 1: Shared user registries

► Scenario 2: Protecting Web resources

► Scenario 3: Tivoli's WebSphere plug-in

► Scenario 4: Using the aznAPI

The scenarios are starting from the easiest level where the integration is not very tight, then it goes into closer integration between the two products.

12.5 Scenario 1: Shared user registries

This scenario will discover, how to share user registries between Tivoli Access Manager V3.9 and WebSphere Application Server V5.0 is in the use of a common user registry. By using the same repository for users, you can share user identities with both Access Manager and WebSphere.

IBM Directory Server
In this section, we will cover the necessary steps to configure WebSphere with IBM Directory Server V3.2.2. IBM Directory Server was installed using the Access Manager ezinstall script for Windows. In addition, we have also installed the Access Manager Policy Server on the same system.

> **Tip:** From a battle scarred veteran
>
> When using the native installation method for Access Manager and installing Access Manager components on the same system as the IBM Directory Server, you must first install the GSKIT component manually. This step is required because the GSKIT component supplied with the IBM Directory Server is at a lower level than the level required for Access Manager. If you do not perform a native installation of the Access Manager GSKIT component before beginning the native installation of the IBM Directory Server, then you will not be able to perform installation of any Access Manager components on that system without uninstalling the IBM Directory Server and all of its components. For details on performing a native installation of the GSKIT component for Access Manager, refer to the *Access Manager for eBusiness V3.9 Base Installation Guide.*
>
> For those of you rereading this section, it is safe to assume that you did not read the above tip the first time. It is Friday night, and while your office co-workers are enjoying happy hour at your favorite local establishment, you are working alone in the lab. So now you are not only working late and missing happy hour, you are also missing out on the latest office gossip, and won't have any idea what anyone is talking about Monday. Take heart, however. I am quite sure that in the future you will pay very close attention to tips in Redbooks, and you won't miss happy hour again.

In order to configure WebSphere for access to the IBM Directory Server, we must first define a user entry for WebSphere to use when binding to the directory. The user entry must also have permission to perform directory-searches on the areas of the ldap tree where WebSphere user and group entries will be stored and allow it to populate these entries in the directory. We are going to create a user entry, wasadmin, for this purpose, using the administration gui, Web Portal Manager, provided with Access Manger.

The Web Portal Manager (WPM) is a Web based GUI that provides the administration of users, groups, and the object space of Access Manager. By using the Web Portal Manager, we can add users to the directory which can be used by both WebSphere and Access Manger in one step.

1. To begin, login with your favorite browser to the Web Portal Manager at https://<your_WPM_server>/pdadmin.

> **Note:** We access the Web Portal Manager using an SSL connection. When the Web Portal Manager is configured, it disables http access to the system; only https connections are allowed.

2. As a next step add a user entry for **wasadmin**. Select **User -> Create** from the navigation bar, then provide the following information:

   User ID: `wasadmin`

   Password and Confirm Password: `password`

   Description: `WAS LDAP admin ID`

   First Name: `WebSphere`

   Last Name: `Administrator`

   Registry UID: `cn=wasadmin,o=itso`

   Check the **Is Account Valid, No Password Policy** and **Is Password Valid** check boxes.

   Once you have finished making the above entries, click the **Create** button. The user will be added to the directory, and WPM will display a confirmation message indicating that the user entry has been created.

3. Now that the wasadmin user is added to the registry, we must set the appropriate access rights for it in the IBM Directory Server. To do this, we use the Directory Management Tool (DMT), which is included in the IBM Directory Server client component. Click the **Add Server** button on the DMT panel, provide the following connection information, then click **OK**.

   Server name: `dirsrv01`

   Port: `389`

   Authentication type: `Simple`

   User DN: `cn=root`

   User password: `password`

4. We will now proceed to add the access rights for the user wasadmin. Select the suffix under which your user entries are defined for WebSpherein this case: **o=itso**.

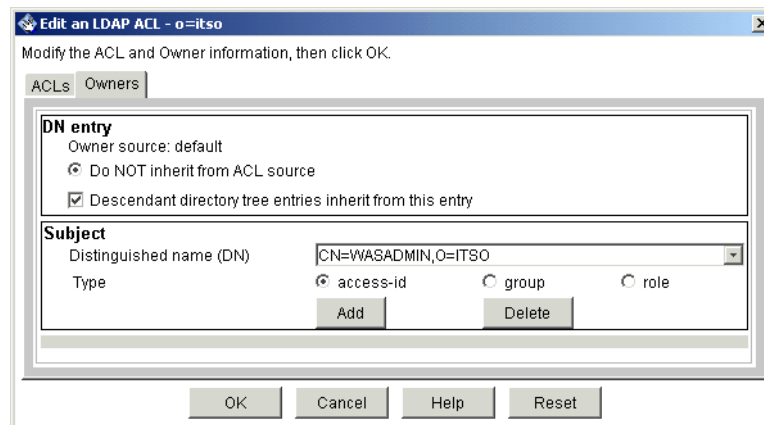5. Click the **ACL** button, the Edit an LDAP ACL panel will display.

Figure 12-3   Edit an LDAP ACL panel

   a. Select the **Owners** tab, and follow the next steps.

   b. Select the **access-id** radio button.

   c. In the Distinguished Name (DN) field enter the DN for wasadmin:
      cn=wasadmin,o=itso.

   d. Click **Add.**

   e. Click **OK**.

The DMT main panel will now be redisplayed. By assigning the wasadmin id
owner authority, we have provided all access rights to the suffix o=itso in our
directory server. If you have multiple suffixes within your directory that contain
WebSphere users, then you will need to repeat the above steps for each suffix
you have defined in your directory.

Configuring WebSphere access to IBM Directory Server
The next step is to configure WebSphere to use the IBM Directory Server as its
user registry.

Follow the steps from Section 10.4.2, "LDAP" on page 245 from the
Administering WebSphere security Chapter. You can follow and use the same
settings that are introduced in the section.

As an additional step the search filter for LDAP search has to be changed
according to the Tivoli Access Manager settings.

1. In the Administrative Console navigate to the **Security -> User Registries ->
   LDAP** item, then select **Advanced LDAP Settings** at the bottom of the page.

2. Modify the User filter field by adding the **(objectclass=inetOrgPerson)** part to reflect the following configuration: (&(uid=%v)**(objectclass=inetOrgPerson)**(objectclass=ePerson))

3. Modify the Group filter field by adding the **(objectclass=accessGroup)** item to reflect the following configuration: (&(cn=%v)(I(objectclass=groupOfNames)**(objectclass=accessGroup)**(objectclass=groupOfUniqueNames)))

4. Modify the Group Member ID Map field by adding the **accessGroup:member** item to reflect the following configuration: groupOfNames:member;**accessGroup:member;**groupOfUniqueNames:uniqueMember

5. Save the configuration for WebSphere. If you are planning to enable SSL for the LDAP connection go ahead and configure it following the instructions form the next section; if not then restart WebSphere to make the changes live.

Configuring WebSphere SSL access to IBM Directory Server

Now that we have WebSphere configured to use the IBM Directory Server, you need to decide whether you need to secure the message traffic between WebSphere and the directory server. Using non-SSL for our connection, all message traffic between WebSphere and the directory server will not be encrypted, meaning that someone could capture the data flowing between WebSphere and the directory, and could find our user ids and their passwords. For a development environment this is probably fine, but once we move our application into a production environment, we may find this to be less than desirable.

During the installation of the IBM Directory Server, using the *ezinstall_ldap_server* script for Access Manager, we chose to enable SSL connections between IBM Directory Server and our Access Manager components to ensure that our message traffic was secure.

1. First you have to configure your WebSphere Application Server's LDAP settings to support SSL for the LDAP connection. Follow the steps from Section , "Configuring secure LDAP (LDAPS) connection" on page 325.

2. In order to provide SSL access between WebSphere and the directory server, we must establish a trusted relationship between them. This requires that WebSphere, when binding to the directory server, must have a means to identify the directory server. We are going to accomplish this by placing the directory servers public certificate into the WebSphere trusted servers keyring file. It is a similar scenario to Section 10.11, "SSL between the Web server and WebSphere" on page 300, and the difference here is that SSL between the LDAP server and WebSphere. IBM Directory Server used the same keystore type, KDB, that IBM HTTP Server uses.

Extract the certificate from the Access Manager's directory server ,IBM Directory Server. During the installation we selected the default option for the keystore; which means that the keystore can be found on the Access Manager server at **C:\keytabs\pd_ldapkey.kdb**. Use the password: **gsk4ikm** to open the keystore; and export the **PDLDAP** certificate from it.

Save the certificate to the **C:\keytabs\ldapcert.arm** file.

You can use the last steps from Section , "Generating a self-signed certificate for the Web server plug-in" on page 301 for help.

3. Import the extracted certificate into WebSphere's server trust file, we were using the Dummy keyfile sets, so import the certificate into the **<WebSphere_root>\etc\DummyServerTrustFile.jks**, the password is **WebAS** to open the Dummy keystore.

> **Note:** If you are not using the Dummy keystore for your LDAP SSL connection, you will have to import the certificate into the Server Trust file of your SSL entry that is used for secure LDAP connection. You can configure this for LDAP using the Administrative Console, under **Security -> User Registries -> LDAP** at the SSL Configuration field.

The entry name when you import the certificate is: **PDLDAP**.

You can use the last steps from Section , "Importing the certificate into the Web Container keystore" on page 304 for help.

4. Once the certificate is imported into WebSphere, the IBM Directory Server must be configured in a way that it can use SSL with 128 bit encryption. Follow the steps from Section , "Configuring the IBM SecureWay Directory Server" on page 327 to do the configuration. The steps are the same for the IBM Directory Server as for the IBM SecureWay LDAP Directory.

5. Stop and restart you WebSphere server. You are now using SSL to communicate between your WebSphere server and the directory server.

12.5.1 Single Sign-On with WebSEAL

When using a reverse proxy such as WebSEAL to authenticate users in the DMZ, it is desireable that WebSphere, as well as other back-end applications and services, trust the authentication that has been performed and the identity that is being presented by the reverse proxy. If this trust can be established, users then need only authenticate once to the reverse proxy in order to have access to all authorized services located behing that proxy. This is commonly known as *Reverse Proxy Single Sign-On*, or RPSS.

There are two ways to establish a trust relationship between WebSphere and WebSEAL:

1. Using *Lightweight Third Party Authentication* (LTPA) tokens
2. Using a *Trust Association Interceptor*(TAI)

Each of these mechanisms of estabilishing trust will be discussed in detail below.

Lightweight Third-Party Authentication

An LTPA token is an encrypted string that contains a userID, an expiration time, and a digital signature. By returning a cookie containing this string (known as an LTPA cookie) to client browsers upon successful authentication, other servers which trust the issuer of the LTPA cookie can request the cookie, read the LTPA token and determine the authenticated userID. The basis of trusting the issurer of the LTPA cookie is that the LTPA token contains the correct digital signature.

> **Note:** The *third party* referred to in the name Lightweight Third-Party Authentication refers to the server which performed the user authentication and issued the LTPA token. *Third party,* therefore, does not refer to the registry being used to authenticate principals. Servers which trust the third party authenticator are said to have *delegated authentication* to this third party. Confusingly, some documents refer to the user registry as the third party in an LTPA environment.

> **Note:** LTPA is an IBM technology that is currently understood only by IBM producs such as WebSphere Application Server, WebSEAL, and Lotus Domino. LTPA has not received industry-wide acceptance.

LTPA does not necessarily require that a reverse proxy be involved in authentication. For example, a user can receive an LTPA cookie from a Domino server after successful authentication, and then use that cookie when communicating with a WebSphere Application Server which trusts the LTPA token issued by the Domino server. In real-world applications, however, authentication is usually performed by a reverse proxy, and the rest of this discussion will assume that scenario, as shown in Figure 12-4.
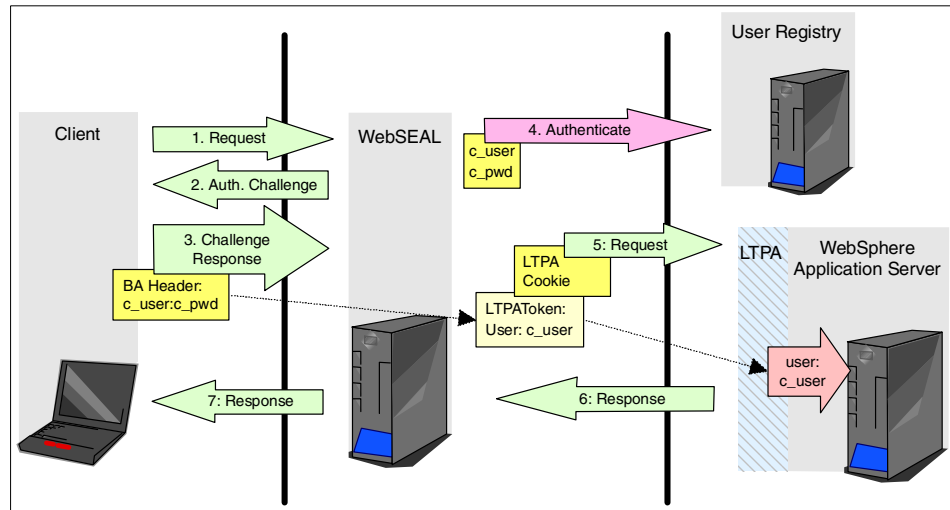
Figure 12-4   LTPA Information flow

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 could be configured to provide a login form instead, but the overall flow of information would remain 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 Authenticaion (BA) Header.

4. WebSEAL authenticates the user against the user registry using c_user and c_pwd.

5. WebSEAL constructs an LTPA token and attaches it to an LTPA cookie which is associated with the request sent to the WebSphere Application Server. WebSEAL can (and should) filter the client's username and password out of the BA Header in the request sent to WebSphere because WebSphere will not need this information. When WebSphere requests the LTPA cookie from WebSEAL, it decrypts the LTPA token and verifies that the signature is correct. Then it trusts that the identity of the request is c_user, as specified in the LTPA token.

6. WebSphere sends output to WebSEAL.

7. WebSEAL sends the output to the client. WebSEAL does not send the LTPA cookie to the client, but rather the cookie is stored in WebSEAL's LTPA cache. This is advantageous since LTPA tokens, if sent to the client over the Internet, could be decrypted over time. Because the LTPA signature never changes,

intercepting LTPA cookies and cracking LTPA tokens would be an easy and effective way to breach an otherwise secure environment.

Configuring an LTPA-enabled WebSEAL Junction

The following procedure will describe the steps necessary to configure WebSphere to trust LTPA tokens that are issued by WebSEAL. This involves generating an LTPA key file on the WebSphere server, copying the key file to the WebSEAL server, and using the key file when configuring the WebSEAL junction.

1. On the WebSphere Administrative Console, click **Security** -> **Authentication Mechanisms** -> **LTPA** to see the LTPA configuration panel, as shown in Figure 12-5.

2. Change the password if necessary.

> **Note:** The first time that security is enabled with LTPA as the authentication mechanism, LTPA keys are automatically generated with the password entered in the panel. In this procedure, however, LTPA keys will be generated manually so that they can be immediately exported and copied to the WebSEAL server.

3. Click the **Generate Keys** button.

4. In the **Key File Name** field, enter the full path of a file on the WebSphere server where the key file should be placed.

5. Click **Export Keys** to create the exported key file. The LTPA key file is a text file which will look something like the one shown in Example 12-1.

Example 12-1

```
#IBM WebSphere Application Server key file
#Thu Aug 15 14:28:47 EDT 2002
com.ibm.Websphere.CreationDate=Thu Aug 15 14\:28\:47 EDT 2002
com.ibm.Websphere.ltpa.version=1.0
com.ibm.Websphere.ltpa.3DESKey=/VrD4i4I8XIiXK6AF/ELOiM9YRgH8IVdp7ji+BJPSDM\=
com.ibm.Websphere.CreationHost=appsrv02
com.ibm.Websphere.ltpa.PrivateKey=5TXFvlm/vs1BNh+fqbrogwve8+NDJzQvlxbzD8i/vRvjT
oWSRrrWQPvBtjKKCv1KBeL/+/RkxsUZdWugVV+SH4WyTL7lIko3P+xjV/B53Ikrdu+fMJOSXm9B3lVM
JtuRPnHxTBQhkKOYgjgkvXw59AbKnaV9gOvhacmzK8OV5DcMngPYpn5eo3mdqVnMw7Oecwr7O53xSdU
5AQ4/O2Yp3NLy7nqrnnOJejgVYy715ekG4k1VjscsPupvykvv8/flDeU8nfInNiiz6K4APtC77SC9mg
MvA3XoEhVxcs3m1KDvH4WvcOo34DK64UDSNiPFDYQ2cmIZZN5Ol7L4BmXOLqBNgmEn+OnR6mbA4sWGv
Sg\=
com.ibm.Websphere.ltpa.Realm=secsrv01\:389
com.ibm.Websphere.ltpa.PublicKey=AKYNppCJNZsoPcC8NmgSo6P8c+zbO8Nm4tGO8EFNOOkjUm
kb6614czbsxN7GAyO/6jr4FkOPf3vWiIe955iLVInE9eLtgJcTFa8dVi9LgKOMKgPkFhO2J/wLcMBJB
PzSJ2A7YtfB/2+gMSRLumptQmIu4eOc3OJTIIAkUcPOdfXpAQAB
```

6. Click **Apply** to accept the changes to the LTPA configuration.

Figure 12-5   LTPA Configuration Panel

7. Under the **Additional Properties** heading, click **Single Signon (SSO)**

8. In the Single Sign-On (SSO) panel, check the **Enabled** box and click **OK** to accept the changes, as shown in Figure 12-6.

Figure 12-6   Single Signon (SSO) panel

9. On the Global Security Panel, change the **Active Authentication Mechanism** to **LTPA (Light weight Third Party Authentication)**.

10. Click **Save** to save the changes to the WebSphere repository.

11. Copy the LTPA key file to the WebSEAL server. Note that this file should be kept very secure, otherwise the LTPA trust relationship may be compromised.

12. Now we can create the junction. In this example, the WebSEAL server's identification is *Webseald-wsl01,* the LTPA key file is located at `c:\keytabs\ltpa.txt,` the LTPA key file password is *password,* the WebSphere server's IP address is 10.30.10.52, and the WebSEAL and WebSphere servers communicate over the junction using SSL. This configuration requires that the root certificate of the CA which signed the WebSphere server's certificatie be added to the WebSEAL certificate keyfile (*pdsvr.kdb).* Using **pdadmin** on the WebSEAL server, execute the followingcommand:

```
server task Webseald-wsl01 create -t SSL -b filter -A -F
"c:\keytabs\ltpa.txt" -Z "password" -h 10.30.10.52 -p 9443 /ltpabank
```

> **Note:** While it is not required that the junction be confiured to use SSL, it is highly recommended unless the channel between WebSEAL and WebSphere is otherwise secured.

Trust Association Interceptor (TAI)

The Trust Association Interceptor feature is another way to establish trust between WebSphere and a reverse proxy in order to achieve Single Sign-On. Rather than relying on a pre-defined token as in the case of LTPA, The Trust Association Interceptor feature defines an API which allows WebSphere to use any available method to validate the input stream.

A trust association interceptor is a Java class which implements the `com.ibm.Websphere.security.TrustAssociationInterceptor` interface, and each implementation of a Trust Association Interceptor is specific to the characteristics of the reverse proxy being used. The interceptor is responsible for validating the request and providing the authenticated userid to the WebSphere security runtime. The WebSphere security runtime then maps the username to a valid LTPA credential that is used internally for authorization purposes. Section 8.4, "Custom Trust Association Interceptor" on page 190 describes the Trust Association Interceptor API in detail. The rest of this section will focus on the WebSEAL Trust Association Interceptor.

The WebSEAL Trust Association Interceptor, running on the WebSphere server, validates the WebSEAL request by authenticating a special user ID and password which is attached to the HTTP Basic Authentication (BA) header of the request.

> **Note:** If the WebSEAL interceptor mutualSSL property is set to **true**, this authentication step is skipped (See below).

After successful authentication of this special userid, the interceptor returns the real client's userid in the IV_USER header.
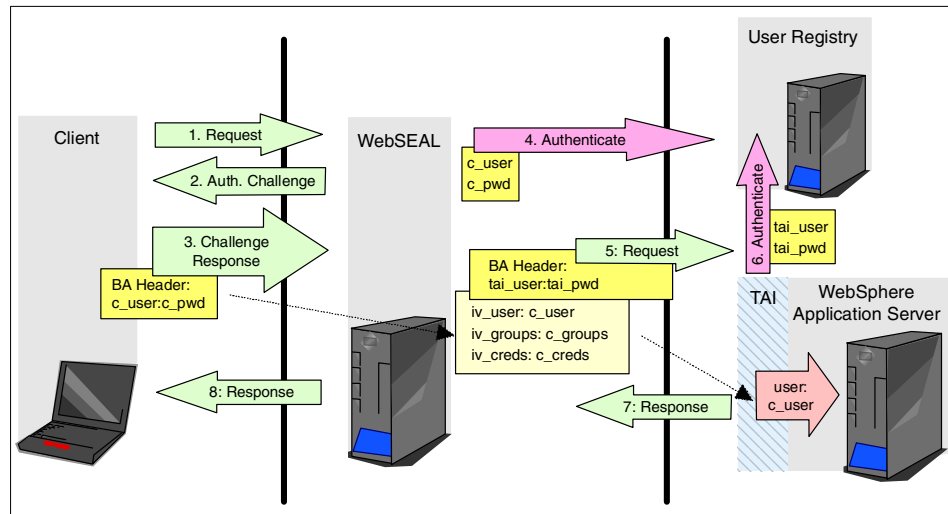

Figure 12-7   TAI Information Flow

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 could be configured to provide a login form instead, but the overall flow of information would remain 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 Authenticaion (BA) Header.

4. WebSEAL authenticates the user against the user registry using c_user and c_pwd.

5. WebSEAL modifies the BA Header so that the userid (tai_user) and password (tai_pwd) are those expected by the Trust Association Interceptor. It attaches

the client's userid and, optionally, group membership and credentials into an additional HTTP headers (iv_user, iv_groups, and iv_creds) that are sent along with the request to WebSphere.

6. WebSphere's Trust Association Interceptor authenticates the userid and password contained in the BA header(tai_user:tai_pwd) in order to establish trust, and then extracts the client's identity (c_user) from the iv_user header. WebSphere then handles the request as comming from c_user.

7. WebSphere sends output to WebSEAL.

8. WebSEAL sends the output to the client.

Configuring a TAI-enabled WebSEAL Junction

The WebSEAL Trust Association Interceptor is configured by assiging values to the properties shown in the table below. The WebSphere server must be restarted whenever the interceptor properties are modified.

Table 12-3   com.ibm.WebSphere.security.Webseal.id properties

Property key	Value
com.ibm.Websphere.security.Webseal.hostname	A comma-delimited list of hostnames from which the interceptor can receive HTTP requests. Requests originating from other hosts will be ignored. If this property is not specified, HTTP requests will be accepted from all hosts.
com.ibm.Websphere.security.Webseal.ports	A comma delimited list of ports from which HTTP requests must originate to be processed. Requests originating from other ports will be ignored. If this property is not specified, HTTP requests will be accepted from all originating ports.

Property key	Value
com.ibm.Websphere.security.Webseal.id	This property specifies the HTTP header elements which *must* be contained in each request received by the interceptor. Requests which do not contain the specified HTTP header elements are ignored by the interceptor. This property must contain either or both of the following values (comma delimited): ▶ iv_user ▶ iv_creds
com.ibm.Websphere.security.Webseal.loginID	This property specifies the userID (e.g. *tai_user* in the scenario above) which will be authenticated, using the password appearing in the HTTP Basic Authentication (BA) header to validate the incomming request. If this property is used, the userID appearing in the BA header is ignored. If this property is *not* used, the interceptor authenticates using both the userID and password appearing in the BA header. This property has no effect when the mutualSSL property is set to **true**.
com.ibm.Websphere.security.Webseal.mutualSSL	If this property is set to **true**, the WebSEAL interceptor implicitly trusts that the WeEAL junction has been secured through the use of one of WebSEAL's mutually authenticated SSL junction capabilities. When this property is set to `true`, the interceptor skips the authentication step in the validation of the request.

> **Important:** Setting the mutualSSL property to **true** effectively disables one of the mechanisms of validating the WebSEAL server and its authentication of the client's identity. In some instances, it may be sufficient for the interceptor to validate the request on the basis of the originating hostname and port, but in general this should be done with caution.

> **Important:** If the interceptor ignores or fails to validate a request, the WebSphere security runtime will proceed to handle the request as if the interceptor had not been enabled. In other words, requests that are not handled by the interceptor are not rejected, but rather are passed unchanged to the security runtime.

It follows that there are three ways of using the HTTP Basic Authentication (BA) headers to validate the WebEAL server and thereby trust its authentication of the client's identity:

1. Send both a userID and password in the BA header. This would be configured by doing the following:

    a. Configure the junction using `-B` `-U` **"***tai_user***"** `-W` **"***tai_pwd***"**

    b. Set the mutualSSL property to **false.**

    c. Do not define the loginID property.

2. Send only a password in the BA header. This would be configured by doing the following:

    a. Set the loginID property to a specical userID (e.g. *tai_user*).

    b. Add the password for *tai_user* to the `basicauth-dummy-passwd` variable in the `Webseald.conf` file.

    c. Configure the junction using `-b supply`,or alternatively use `-B` `-U` **"***dummy_userID***"** `-W` **"***tai_pwd***"**. The *dummy_userID* value in the BA header will be ignored.

3. Send nothing in the BA header. This would be configured b doping the following:

    a. Set the mutualSSL property to **true**

    b. Enable the junction to be a mutually authenticated SSL junction by using a client certificate to authenticate WebSEAL to the back-end server.

The following procedure details the steps necessary for creating a WebSEAL junction using Single Sign-On based on the Trust Association Interceptor. The interceptor will be configured to validate the request by authenticating a special userID and password supplied in the BA header by WebSEAL.

Create a userid in the registry that the interceptor will use to validate the request. In this example we will assume the userid `tai_user` with passoword `tai_pwd` has been created in the registry.

Configure WebSphere to use TAI

The following steps will show, how to configure WebSphere to use TAI for authentication.

1. On the WebSphere Administrative Console, click **Security -> Authentication Mechanisms -> LTPA** to view the LTPA configuration panel.

2. Under **Additional Properties**, click **Trust Association** to see the Trust Association Panel shown in Figure 12-8
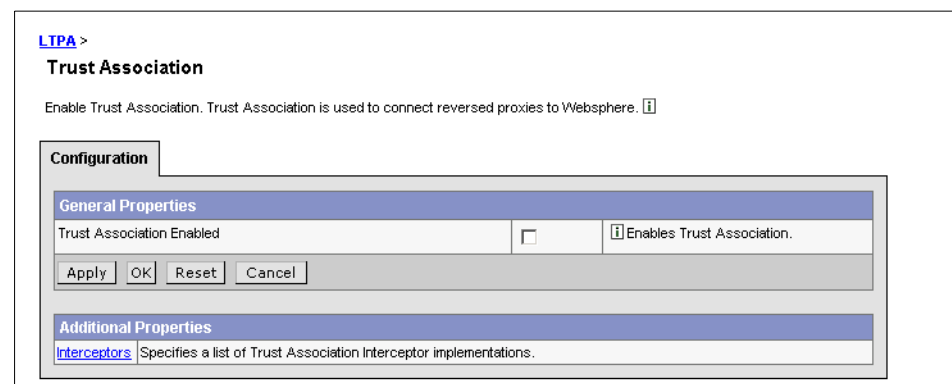
Figure 12-8   Trust Association Panel

3. Click the **Trust Association Enabled** checkbox to enable Trust Association

4. Click **OK** and then click **Interceptors** in the Additional Properties section.

5. On the Interceptors panel, click the WebSEAL interceptor, **com.ibm.ws.security.Web.WebSealTrustAssociationInterceptor**

6. On the interceptor configuration pnel, verify that the Interceptor Classname field contains the value:
   `com.ibm.ws.security.Web.WebSealTrustAssociationInterceptor`

7. The next step is to configure properties for the WebSEAL interceptor. click **Custom Properties** under **Additional Properties.**

> **Note:** There are two ways to pass properties to the interceptor: By using a properties file, and by seting custom properties in the WebSphere Administrative Console. At the time of this writing, the properties file was not fully implemented, so the custom properties mechanism was used instead.

8. On the Interceptor Custom Properties panel, create the following key and value pairs. In this example, it is assumed that the WebSEAL server 's hostname is *wsl01*, and that the WebSEAL server will be communicating with the client browser over port 443.

   – com.ibm.Websphere.security.Webseal.id = **iv_user**

   – com.ibm.Websphere.security.Webseal.hostnames = **wsl01**

   – com.ibm.Websphere.security.Webseal.ports = **443**

   – com.ibm.Websphere.security.Webseal.loginId = **tai_user**

   – com.ibm.Websphere.security.Webseal.mutualSSL = **false**

9. Now we can create the junction. In this example, the WebSEAL server's identification is *Webseald-wsl01,* Using `pdadmin` on the WebSEAL server, execute the followingcommand:

```
server task Webseald-wsl01 create -t SSL -B -U"tai_user" -W"tai_pwd" -h
10.30.10.52 -p 9443 /taibank
```

Configuring TAI with WebSEAL

In this scenario, we will be configuring the Access Manager for eBusiness V3.9 WebSEAL component to connect to our WebSphere server using TAI. To perform the configuration, follow the steps below.

1. To begin configuring WebSphere for TAI support, start the WebSphere Administrators console, and login. Once the Administrators console main panel is displayed, on the left pane, select **Security -> Authentication Mechanisms -> LTPA**. The following panel will then display.

Figure 12-9   WebSphere LTPA panel

2. We will first configure LTPA authentication. To do so, enter a **password** to use with LTPA in the Password and Confirm Password fields, and then click the **OK** button. The Global Security panel will then display. Return to the LTPA panel, and select **Trust Association**. The following panel will then be displayed.

Figure 12-10   WebSphere Trust Association panel

3. Check the **Trust Association Enabled** box, and then click the **OK** button. You will be returned to the LTPA panel. Select **Trust Association** again, and

return to this panel. Select the **Interceptors** link, the Interceptors panel will then be displayed.

4. On the Interceptors panel, select the WebSEAL interceptor, **com.ibm.security.Web.WebSealTrustAssociationInterceptor**. The com.ibm.security.Web.WebSealTrustAssociationInterceptor panel will then be displayed.

5. On the WebSealTrustAssociationInterceptor panel, click **Custom Properties**. From this panel, we will define the properties for our WebSEAL TAI connection.

> **Note:** These properties can also be defined in a properties file.

6. In this example, we will define the trust association properties using custom properties. To begin, click the **New** button.

7. From this panel, we will define our properties. In the Name field, enter **com.ibm.Websphere.security.Webseal.id**. This property defines the HTTP header value WebSphere is to use to obtain the actual user id for this request. In the Value field, enter **iv-user**. If you wish, you may also enter a description in the Description field. When done, select **OK**.

8. Repeat the previous two steps and add the following properties:

Table 12-4   Trust Association Interceptor properties

Property	value
com.ibm.Websphere.security.Webseal.id	iv-user
com.ibm.Websphere.security.Webseal.hostname	wsl01 the name of the WebSEAL server
com.ibm.Websphere.security.Webseal.ports	443 Because we are using an SSL connection

> **Tip:** From a battle scarred veteran
>
> WebSphere, when receiving a connection request over a TAI connection, uses this value when validating security credentials. To do this, it uses only the hostname of the requestor, and not the fully qualified DNS name. If you enter the full DNS name here, you will find that the request will not be processed by WebSphere TAI, and the user identity used for the request will be that of the WebSEAL server. This is probably not the result you wish to achieve.
>
> If you chose yet again to ignore my advice, then I must assume that you really like the company of your lab machines.

9. Once you have completed entering your properties, on the left pane, select **Security -> Global Security**.

10. The **Enabled** box should already be selected. If you have not yet configured security for your WebSphere Server, you should go back and do so now, before continuing. Assuming you already have security enabled, scroll down the right pane, and in the Active Authentication Mechanism field, select **LTPA**. Select the **OK** button, and save your configuration.

11. Finally restart your WebSphere server to continue.

Configure the WebSEAL Server

Now that you have WebSphere configured for TAI support to WebSEAL, we must get our WebSEAL server set up and configured. To do so, we must first set up our trust association between WebSEAL and WebSphere. To do this, follow these next steps.

1. In our example, we are using the sample keyring files installed with WebSphere. Note that if you have obtained certificates for your WebSphere server, you may skip this step, and proceed to step 5 below to import the signer certificate for WebSEAL. To begin, start the ikeyman utility for WebSphere, and open the server key file for WebSphere. Open the **DummyServerKeyFile.jks** in the <WebSphere_root>\etc directory. You will be prompted for the keyfile password; if using the dummy file, this is WebAS. The IBM Key Management panel will then be displayed. Select **Personal Certificates**, and the following panel will be displayed.

2. We now need to extract our WebSphere certificate. Select the **Websphere dummy server** certificate, and then click the **Export/Import...** button. The following panel will then be displayed.

3. For Data type, select **Base64-encoded ASCII data**. For Certificate file name, enter the file name for the certificate. In our example, we used **WebSphereServerCert.arm**. In the Location field, enter the path to the directory you wish to store the certificate in. In our example, we stored the

certificate in **etc** directory of the WebSphere root. Once you have completed your entries, click the **OK** button.

4. Once you have saved your certificate, you will need to transfer it to your WebSEAL server. Note that if you have defined your own keyfiles for WebSphere, and have obtained a certificate from a CA, that you can use the root CA's certificate which signed your WebSphere certificate in the following steps instead.

5. To set up the trust relationship between WebSphere and WebSEAL, we now need to import the certificate just saved, or the signing root CA certificate for our WebSphere server. This will establish a trust relationship for WebSEAL. Note that if you wish to use mutual SSL authentication, you will also need to setup the trust relationship for WebSphere in a similar matter.

6. To begin, on your WebSEAL server, start the GSKIT GUI. Open your WebSEAL key database. We are using the WebSEAL default database, and selected **<WebSEAL_root>\www\certs\pdsrv.kdb**. You will then be prompted for the key database password. The password for the default WebSEAL database is **pdsrv**. Once the database is opened, select **Signer Certificates**. The following panel will then be displayed.

7. Select the **Add...** button. The following panel will then be displayed.

8. For Data type, select **Base64-encoded ASCII data**. In the Certificate file name field, enter the name of the certificate. In our example, this is **WebSphereServerCert.arm**. In the Location field, enter the path to the directory that you have stored your certificate file. In our example, we entered **<WebSphere_root>\etc**. Once you have finished your entries, select the **OK** button. You will then be prompted for a label name to store your certificate with. Enter a name that will make it easy for you to identify that this certificate is for your WebSphere server. The IBM Key Management panel will then be displayed, and the certificate you just added will now be displayed as the label name you specified. You may now close the IBM Key Management utility. We now have established the trust relationship for our WebSEAL server.

Now that we have configured WebSphere for TAI support, and we have set up the trust relationship for our WebSEAL server, we can now define our SSL junction between WebSEAL and WebSphere. Note that when using TAI, you must define an SSL junction. TAI support is not provided by WebSphere on a non-SSL connection.

We can now define a junction for our WebSEAL server to connect to WebSphere using TAI. Create a user id and password in the user registry you are using for WebSphere, for example: `tai_user` with the password: `tai_pwd`. Issue the following command, with the right parameters for your environment, in pdadmin to create your junction:

```
server task Webseald-WebSEALServer create -t SSL -c iv_user -B -U "WebSEALid"
-W "WebSEALpassword" -h WebSphereServerName -p SSLport /JunctionName
```

► WebSEALServer: the hostname of your WebSEAL server, for example: `wsl01`.

► WebSEALid: the user id you have created for your WebSEAL server. Note that if you have set the com.ibm.Websphere.security.Webseal.loginid, then you should specify a dummy id, and not the actual WebSEAL user id.

► WebSEALpassword: the password for your WebSEAL server.

► WebSphereServerName: the hostname of your WebSphere server, for example: `appsrv01`.

► SSLport: the port number defined in WebSphere for SSL connections, for exmaple: `9443`.

► JunctionName: the name for this junction, for example: `/tai`.

After defining your junction, you will now be able to connect to WebSphere from WebSEAL. When you login to WebSEAL, and access your WebSphere server over your TAI junction, the Access Manager user id will be passed to, and used by, WebSphere when invoking your application. Your users will no longer see a basic authentication challenge from your application; instead, the user credentials passed by WebSEAL over the TAI junction will be used by WebSphere, and your users will not have to perform a second login.

If you are reading on, something has probably gone wrong, and it appears that your TAI junction is not working. Never fear, your battle scarred veteran is here to lend a hand. We will need to turn on tracing in WebSphere to narrow down the problem. To begin, we will need to enable tracing, in order to perform this step, follow the instructions at Section , "Security trace" on page 235.

Once your server has restarted, the first thing to look at is to see if TAI is actually enabled. In the example below we have included a portion of the trace file.

Example 12-2   WebSphere Security initialization

```
8/22/02 7:40:44:482 CDT]  7822e45 SASRas        A JSAS0001I: Security
configuration initialized.
[8/22/02 7:40:44:723 CDT]  7822e45 SASRas        A JSAS0002I: Authentication
protocol: CSIV2/IBM
[8/22/02 7:40:44:735 CDT]  7822e45 SASRas        A JSAS0003I: Authentication
mechanism: LTPA
[8/22/02 7:40:44:747 CDT]  7822e45 SASRas        A JSAS0004I: Principal name:
dirsrv01.itso.ral.ibm.com:389/wasadmin
[8/22/02 7:40:44:925 CDT]  7822e45 SASRas        A JSAS0005I: SecurityCurrent
registered.
[8/22/02 7:40:44:938 CDT]  7822e45 SASRas        A JSAS0006I: Security
connection interceptor initialized.
```

```
[8/22/02 7:40:45:014 CDT]   7822e45 SASRas         A JSAS0007I: Client request
interceptor registered.
[8/22/02 7:40:45:192 CDT]   7822e45 SASRas         A JSAS0008I: Server request
interceptor registered.
[8/22/02 7:40:45:509 CDT]   7822e45 SASRas         A JSAS0009I: IOR interceptor
registered.
```

During the initialization for WebSphere security, we should see that LTPA is
being used as the Authentication mechanism, as above. If in fact SWAM is being
used, this is our problem. In this case, we need to go back and check our LTPA
configuration, and on the Global Security panel, ensure that LTPA has been
selected as the Authentication mechanism.

If it appears that LTPA is properly defined, then the next thing we need to check
is that TAI is being initialized. The following portion of our trace file illustrates
what we should see next.

Example 12-3   Trace of TrustAssociation initialization

```
7822e45 TrustAssociat > initialize
[8/22/02 7:41:32:021 CDT]   7822e45 TrustAssociat A SECJ0121I: Trust Association
Init class com.ibm.ws.security.Web.WebSealTrustAssociationInterceptor loaded
successfully
[8/22/02 7:41:32:034 CDT]   7822e45 TrustAssociat d Trust Properties=
                                    {com.ibm.Websphere.security.Webseal.ports=443,
com.ibm.Websphere.security.Webseal.hostnames=wsl01,
com.ibm.Websphere.security.Webseal.id=iv-user}
[8/22/02 7:41:32:035 CDT]   7822e45 WebSealTrustA > Initializing
WebSealTrustAssociationInterceptor...
[8/22/02 7:41:32:035 CDT]   7822e45 WebSealTrustA > getElements
[8/22/02 7:41:32:035 CDT]   7822e45 WebSealTrustA < getElements
[8/22/02 7:41:32:035 CDT]   7822e45 WebSealTrustA > getElements
[8/22/02 7:41:32:035 CDT]   7822e45 WebSealTrustA < getElements
[8/22/02 7:41:32:035 CDT]   7822e45 WebSealTrustA > getElements
[8/22/02 7:41:32:036 CDT]   7822e45 WebSealTrustA < getElements
[8/22/02 7:41:32:036 CDT]   7822e45 WebSealTrustA d WebSeal Login ID = null
[8/22/02 7:41:32:045 CDT]   7822e45 WebSealTrustA > addASource
[8/22/02 7:41:32:045 CDT]   7822e45 WebSealTrustA d WebTAInterceptor: Added
source = wsl01:443
[8/22/02 7:41:32:045 CDT]   7822e45 WebSealTrustA < Exiting addASource
[8/22/02 7:41:32:045 CDT]   7822e45 WebSealTrustA < Exiting initialization:
SUCCESS
[8/22/02 7:41:32:045 CDT]   7822e45 TrustAssociat A SECJ0122I: Trust Association
Init Interceptor signature: WebSeal Interceptor Version 1.1
[8/22/02 7:41:32:064 CDT]   7822e45 TrustAssociat A SECJ0120I: Trust Association
Init loaded 1 interceptor(s)
[8/22/02 7:41:32:076 CDT]   7822e45 TrustAssociat < initialize
```

Here we see TAI being initialized. We need to look at the Trust Properties line in the trace to verify that the property values being used are the ones we think we set. If we see an error here with one of the properties, then we need to go back to our property definition, either in the properties file, or our custom property entries in TAI, and provide the correct values. If the property values are correct, and we see, as above, that the WebSealTrustAssociation is being initialized properly, then we know that we have properly configured TAI. In this case, we now need to access our WebSEAL junction from a browser, and login to WebSEAL. The next example shows what should appear in the trace file if all is well.

Example 12-4   Trace file http header from WebSEAL

```
[8/22/02 7:42:44:163 CDT] 277a2e5c EJSWebCollabo d Http Header names and
values:
authorization=[Basic d2Vic2VhbHM6cGFzc3dvcmQx]
iv-groups=["managergrp"]
via=[HTTP/1.1 wsl01:443]
user-agent=[Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)]
host=[seccli.itso.ral.ibm.ibm.com:9443]
accept=[image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*]
connection=[close]
accept-language=[en-us]
iv-user=[manager]
iv-creds=[Version=1,
BAKs3DCCA-YMADCCA-AwggPsAgIDkDBsMCkwHgIE48b7FgIDAKe5AgIR1gICAKUCASQEBgBAqsIqTAw
...
OYTFCUlowRkJRVUU5BAA=]
referer=[https://wsl01.itso.ral.ibm.com/tai/itsobank/]
accept-encoding=[gzip, deflate]
cookie=[msp=2; IV_JCT=/tai]
[8/22/02 7:42:44:163 CDT] 277a2e5c EJSWebCollabo d VirtualHost is :
default_host
[8/22/02 7:42:44:163 CDT] 277a2e5c WebSecurityCo > WebAccessContext
[8/22/02 7:42:44:163 CDT] 277a2e5c WebSecurityCo < WebAccessContext
[8/22/02 7:42:44:163 CDT] 277a2e5c WebCollaborat >
SetUnauthenticatedCredIfNeeded
[8/22/02 7:42:44:163 CDT] 277a2e5c WebCollaborat d Invoked and received
Credential are null, setting it anonymous/unauthenticated.
[8/22/02 7:42:44:163 CDT] 277a2e5c WebCollaborat <
SetUnauthenticatedCredIfNeeded:true
[8/22/02 7:42:44:164 CDT] 277a2e5c EJSWebCollabo d  Request Context
Path=/itsobank,  Servlet Path=/,  Path Info=transfer/branchtransfer.html
[8/22/02 7:42:44:164 CDT] 277a2e5c WebCollaborat > authorize
[8/22/02 7:42:44:164 CDT] 277a2e5c WebCollaborat d URI requested:
/transfer/branchtransfer.html
[8/22/02 7:42:44:164 CDT] 277a2e5c WebAppCache   d Okay, I found the entry for
[default_host:/itsobank]
[8/22/02 7:42:44:164 CDT] 277a2e5c WebAccessCont > WebAccessContext
```

In this example, the first portion shows a snapshot of the HTTP header of the request received by WebSphere from WebSEAL. Here, we need to look at iv fields being passed to WebSphere from WebSEAL. In our example, we used the **-c all** junction option in WebSEAL. This means that we should see all of the iv header fields being passed to WebSphere from WebSEAL. In our example, you can in fact see that the iv fields were passed. For iv-user, we have manager, which is our Access Manager/WebSphere user id, and is the id with which we want to use in our itsobank application. In addition, you can see that iv-creds and iv-groups were also passed. If, when looking at the HTTP header trace entry, you do not see the iv field which you have configured TAI to use in WebSphere, then the problem is probably in the creation of your WebSEAL junction. In this case, you should go back and redefine your WebSEAL junction, using the correct parameter that you require with the **-c** option.

The next set of trace entries show the invocation of TAI to obtain the user credentials passed by WebSEAL, and the authentication of the WebSEAL server. Note that only the relevant portions are presented.

Example 12-5   WebSEAL Trust Association trace

```
8/22/02 7:42:45:223 CDT] 277a2e5c WebAuthentica d A cookie was received. The
name is LtpaToken and the value is NULL
[8/22/02 7:42:45:223 CDT] 277a2e5c WebAuthentica < handleSSO: (null)
[8/22/02 7:42:45:514 CDT] 277a2e5c WebAuthentica d handleTrustAssociation
[8/22/02 7:42:45:515 CDT] 277a2e5c WebAuthentica d TrustAssociation is enabled.
[8/22/02 7:42:45:586 CDT] 277a2e5c TrustAssociat > getInterceptor
[8/22/02 7:42:45:586 CDT] 277a2e5c TrustAssociat d Check if target interceptor
...
[8/22/02 7:42:46:397 CDT] 277a2e5c WebSealTrustA > getCheckID
[8/22/02 7:42:46:397 CDT] 277a2e5c WebSealTrustA < getCheckID
[8/22/02 7:42:46:397 CDT] 277a2e5c WebSealTrustA d isTargetInteceptor: header
name=authorization
[8/22/02 7:42:46:397 CDT] 277a2e5c WebSealTrustA d isTargetInteceptor: header
name=iv-groups
[8/22/02 7:42:46:397 CDT] 277a2e5c WebSealTrustA d isTargetInteceptor: header
name=via
[8/22/02 7:42:46:398 CDT] 277a2e5c WebSealTrustA d isTargetInterceptor:
VIA=HTTP/1.1 wsl01:443
[8/22/02 7:42:46:515 CDT] 277a2e5c WebSealTrustA > checkVia for wsl01:443
[8/22/02 7:42:46:516 CDT] 277a2e5c WebSealTrustA < getCheckID:  0
[8/22/02 7:42:46:516 CDT] 277a2e5c WebSealTrustA d isTargetInteceptor: header
name=user-agent
[8/22/02 7:42:46:516 CDT] 277a2e5c WebSealTrustA d isTargetInteceptor: header
name=host
[8/22/02 7:42:46:516 CDT] 277a2e5c WebSealTrustA d isTargetInteceptor: header
name=accept
[8/22/02 7:42:46:516 CDT] 277a2e5c WebSealTrustA d isTargetInteceptor: header
name=connection
```

```
[8/22/02 7:42:46:516 CDT] 277a2e5c WebSealTrustA d isTargetInteceptor: header
name=accept-language
[8/22/02 7:42:46:516 CDT] 277a2e5c WebSealTrustA d isTargetInteceptor: header
name=iv-user
[8/22/02 7:42:46:516 CDT] 277a2e5c WebSealTrustA d isTargetInteceptor: header
name=iv-creds
[8/22/02 7:42:46:516 CDT] 277a2e5c WebSealTrustA d isTargetInteceptor: header
name=referer
[8/22/02 7:42:46:517 CDT] 277a2e5c WebSealTrustA d isTargetInteceptor: header
name=accept-encoding
[8/22/02 7:42:46:517 CDT] 277a2e5c WebSealTrustA d isTargetInteceptor: header
name=cookie
[8/22/02 7:42:46:517 CDT] 277a2e5c WebSealTrustA d Yes, it is via WebSeal.
[8/22/02 7:42:46:517 CDT] 277a2e5c WebAuthentica d A TrustAssociation
interceptor is available for this request.
[8/22/02 7:42:46:838 CDT] 277a2e5c WebSealTrustA > Entering
validateEstablishedTrust...
[8/22/02 7:42:46:878 CDT] 277a2e5c WebSealTrustA d Going to authenticate
tai_user.
[8/22/02 7:42:46:973 CDT] 277a2e5c WebAuthentica > basicAuthenticate
...
[8/22/02 7:42:47:448 CDT] 277a2e5c ltpaLoginModu > login()
[8/22/02 7:42:47:448 CDT] 277a2e5c CredentialsHe > copyCredToken(credToken)
[8/22/02 7:42:47:448 CDT] 277a2e5c CredentialsHe d credential token is null
[8/22/02 7:42:47:448 CDT] 277a2e5c CredentialsHe < copyCredToken(credToken)
[8/22/02 7:42:47:448 CDT] 277a2e5c CredentialsHe > copyCredToken(credToken)
[8/22/02 7:42:47:448 CDT] 277a2e5c CredentialsHe d credential token is null
[8/22/02 7:42:47:448 CDT] 277a2e5c CredentialsHe < copyCredToken(credToken)
[8/22/02 7:42:47:448 CDT] 277a2e5c ltpaLoginModu d uid = tai_user
[8/22/02 7:42:47:448 CDT] 277a2e5c ltpaLoginModu d realm = null
[8/22/02 7:42:47:448 CDT] 277a2e5c ltpaLoginModu d password = XXXXXXXX
[8/22/02 7:42:47:448 CDT] 277a2e5c Util          > toString(array)
[8/22/02 7:42:47:449 CDT] 277a2e5c Util          d array is null
[8/22/02 7:42:47:449 CDT] 277a2e5c Util          < toString(array)
[8/22/02 7:42:47:449 CDT] 277a2e5c ltpaLoginModu d cred token = <null>
[8/22/02 7:42:47:449 CDT] 277a2e5c ltpaLoginModu d Successfully gathered
authentication information
[8/22/02 7:42:47:449 CDT] 277a2e5c ltpaLoginModu d Using uid and password for
authentication
[8/22/02 7:42:47:449 CDT] 277a2e5c ltpaLoginModu d Authenticating
"null/tai_user"
[8/22/02 7:42:47:449 CDT] 277a2e5c LTPAServerObj > authenticate
[8/22/02 7:42:47:449 CDT] 277a2e5c LTPAServerObj < authenticate
[8/22/02 7:42:47:449 CDT] 277a2e5c UserRegistryI > checkPassword
[8/22/02 7:42:47:449 CDT] 277a2e5c LdapRegistryI > checkPassword
[8/22/02 7:42:47:449 CDT] 277a2e5c LdapRegistryI d Authenticating
                              tai_user
[8/22/02 7:42:47:450 CDT] 277a2e5c LdapRegistryI d Searching for users
[8/22/02 7:42:47:450 CDT] 277a2e5c LdapRegistryI > getUsers
```

```
                                        tai_user
[8/22/02 7:42:47:450 CDT] 277a2e5c LdapRegistryI > search
[8/22/02 7:42:47:450 CDT] 277a2e5c LdapRegistryI d DN: o=itso
[8/22/02 7:42:47:450 CDT] 277a2e5c LdapRegistryI d Search scope: 2
[8/22/02 7:42:47:450 CDT] 277a2e5c LdapRegistryI d Filter:
(&(uid=tai_user)(objectclass=inetOrgPerson))
...
[[8/22/02 7:42:47:453 CDT] 277a2e5c LdapRegistryI d Found user
                                 cn=tai_user,o=itso
[8/22/02 7:42:47:453 CDT] 277a2e5c LdapRegistryI > checkStopped
[8/22/02 7:42:47:453 CDT] 277a2e5c LdapRegistryI < checkStopped
[8/22/02 7:42:47:486 CDT] 277a2e5c LdapRegistryI d Time elapsed to open/close
DirContext: 33
[8/22/02 7:42:47:486 CDT] 277a2e5c LdapRegistryI d Authenticated with
                                 cn=tai_user,o=itso
[8/22/02 7:42:47:486 CDT] 277a2e5c LdapRegistryI < checkPassword
                                 cn=tai_user,o=itso
[8/22/02 7:42:47:486 CDT] 277a2e5c UserRegistryI d user cn=tai_user,o=itso
password checks ok
```

In this section of the trace, we see that TAI is processing the header information provided by our WebSEAL server, and is authenticating the WebSEAL server, using the user id and password provided with the **-B** option. If the id or password you set in your junction is invalid, this will show up as an authentication error. In this example, the id passed to WebSphere is tai_user, and WebSphere was able to successfully authenticate the WebSEAL server.

In this final section of the trace, once WebSphere has authenticated our WebSEAL server, the user identity passed by WebSEAL will be used for this request. In our example, the user ID passed is **manager**. WebSphere will locate the user ID passed in the user registry, and then use this identity to process the user request. If for some reason the user id passed by WebSEAL is not contained in the user registry being used by WebSphere, you will instead see an error. In this case, you need to check your WebSphere user registry to determine why that user ID can not be found.

Example 12-6   WebSphere security trace for TAI

```
8/22/02 7:42:49:791 CDT] 277a2e5c Authenticatio d
publicName:dirsrv01.itso.ral.ibm.com:389/tai_user
[8/22/02 7:42:49:791 CDT] 277a2e5c Authenticatio d
realm:dirsrv01.itso.ral.ibm.com:389;userName:tai_user
[8/22/02 7:42:49:791 CDT] 277a2e5c Authenticatio d
accessId:user:dirsrv01.itso.ral.ibm.com:389/cn=tai_user,o=itso
[8/22/02 7:42:49:792 CDT] 277a2e5c WebAuthentica < basicAuthenticate
[8/22/02 7:42:49:792 CDT] 277a2e5c WebSealTrustA d Successful authentication
for validateEstablishedTrust.
```

```
[8/22/02 7:42:49:792 CDT] 277a2e5c WebAuthentica d TrustAssociation has been
validated successfully.
[8/22/02 7:42:50:002 CDT] 277a2e5c WebSealTrustA > getAuthenticatedUsername
[8/22/02 7:42:50:006 CDT] 277a2e5c WebSealTrustA < Exiting
getAuthenticatedUsername: manager
[8/22/02 7:42:50:006 CDT] 277a2e5c WebAuthentica d Username retrieved is
[manager]
[8/22/02 7:42:50:006 CDT] 277a2e5c WebAuthentica d Map credentials for manager.
[8/22/02 7:42:50:074 CDT] 277a2e5c SecurityServe > mapCredential
[8/22/02 7:42:50:074 CDT] 277a2e5c SecurityServe d Credential is a Trusted
Credential
...
[[8/22/02 7:42:50:075 CDT] 277a2e5c Credential    < getUserName() -> manager
[8/22/02 7:42:50:075 CDT] 277a2e5c UserRegistryI > createCredential
                                manager
[8/22/02 7:42:50:076 CDT] 277a2e5c LdapRegistryI > createCredential
                                manager
[8/22/02 7:42:50:076 CDT] 277a2e5c LdapRegistryI > getUserDisplayName
                                manager
[[8/22/02 7:42:51:073 CDT] 277a2e5c LdapRegistryI d Found user
                                cn=manager,o=itso
...
[8/22/02 7:42:51:654 CDT] 277a2e5c LTPAValidatio d LTPAValidationCache (cache
enabled): validation = 181 millis
[8/22/02 7:42:51:654 CDT] 277a2e5c Authenticatio > extractCredentialAttributes
[8/22/02 7:42:51:654 CDT] 277a2e5c Authenticatio d
publicName:dirsrv01.itso.ral.ibm.com:389/manager
[8/22/02 7:42:51:654 CDT] 277a2e5c Authenticatio d
realm:dirsrv01.itso.ral.ibm.com:389;userName:manager
[8/22/02 7:42:51:654 CDT] 277a2e5c Authenticatio d
accessId:user:dirsrv01.itso.ral.ibm.com:389/cn=manager,o=itso
[8/22/02 7:42:51:655 CDT] 277a2e5c WebAuthentica < validate
[8/22/02 7:42:51:655 CDT] 277a2e5c WebAuthentica d Mapped credential for
TrustAssociation was validated successfully.
[8/22/02 7:42:51:655 CDT] 277a2e5c WebAuthentica < handleTrustAssociation: OK
[8/22/02 7:42:51:655 CDT] 277a2e5c WebAuthentica d Successful authentication
[8/22/02 7:42:51:700 CDT] 277a2e5c WebAuthentica > createCookie LtpaToken
QNu+31OoJ9pOIKcC+IcAuAubI5rFE4JedMHq2YlKJVOcQsNWkC
```

12.5.2 Forms Authentication Single Sign-On

With the Single Sign-On solution provided with Access Manager, you have the ability to provide your users with the capability to access your WebSphere applications transparently, without them being aware that Access Manager is handling authentication for them to your applications. Up until now, if your existing applications require the use of a login form to authenticate, it was still

necessary for your users to login again to your applications, after performing a login to Access Manager. Howerver, with the release of V3.9 of Access Manager for eBusiness, it is now possible to acheive a Single Sign-On solution to your applications, even for those applications which require the use of a login form.

Tivoli Access Manager for eBusiness V3.9 provides support to login an Access Manager user to a WebSphere application using HTML forms for authentication. When you enable Single Sign-On forms authentication, the WebSEAL component of Access Manager will intercept the login form from your WebSphere application, and will supply the authentication information required back to the application. Your back end WebSphere application will be unaware that the response is from WebSEAL, and not the user, and the user will be unaware that a second login occured.

To enable Single Sign-On forms authentication to a backend application, the Access Manager administrator must do two things. First, 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 second, a junction must be created to the back end Web server using the **-S** option, which specifies the location of the configuration file. Once this is completed, WebSEAL will provide login support for Access Manager users to the back end WebSphere application.

For further information on enabling single-sign on forms authentication, refer to the *Access Manager for eBusiness WebSEAL Administrators Guide*.

Creating the Single Sign-On forms authentication configuration file
The purpose of the configuration file for Single Sign-On forms authentication is to define the following 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.

Consider our sample application, ITSOBank sample application. It requires that a user login, using a login form. Below is the HTML source for our login page.

Example 12-7   ITSOBank sample application login form

```
<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"> <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 which WebSEAL will need to fill in. The next example shows the
Single Sign-On forms configuration file we created.

Example 12-8   Single Sign-On 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 = was50
argument-stanza = args-for-login-itsobank
[args-for-login-itsobank]
j_username = gso:username
j_password = gso:password
```

In this example, we have configured one login form page, login-itsobank. The
URI for our login form is `/itsobank/login/login.html`. This entry defines for
WebSEAL the URI that should be intercepted. When a request is received for
this URI, WebSEAL will intercept the form, and will return to our ITSOBank
application the GSO user ID and password defined for this Acess Manager user
in the `was50` GSO resource. If the user does not have a GSO ID and password
defined for was50, then WebSEAL will return an error page to the user to inform
them that they can not login to the itsobank application.

We now need to create a junction to our back end WebSphere server, using the
-**S** parameter. Once we have done this, Single Sign-On forms authentication will
be enable. The syntax of the junction command is:

```
pdadmin> server task Webseald-WebSEALServer create -f -t tcp -p portnumber -h
WebSphereServerName -S path/filename.conf /JunctionName
```

Where the following arguments are defined:

- WebSEALServer is the host name of your WebSEAL server, for example: `ws101`.
- portnumber is the port number to connect to WebSphere, for example: `9443`.
- WebSphereServerName is the host name of your WebSphere server, for example: `appsrv01`.
- path/filename.conf is the full path and name of your configuration file.
- JunctionName is the name you with to assign to this junction, for example: `/tai`.

After creating your junction, any request which causes the itsobank application to present the login.html form will be intercepted by WebSEAL, and WebSEAL will provide the users id and password back to the ITSOBank sample application. The end user will never be aware that a login to ITSOBank sample application was performed on his behalf.

12.5.3 Tivoli Access Manager plug-in for WebSphere Edge Server

The WebSphere Edge Server is a collection of applications designed to improve Web and application server performance and availability by load balancing servers, intelligently caching static content, and by moving content delivery as close to the users, from a network perspective, as possible. The "edge of the network" is normally the DMZ between an organization's intranet and the public Internet, and it is into this DMZ that the WebSphere Edge Server components are deployed.

The Edge Server Caching Proxy component can act as a reverse proxy, very similar to the WebSEAL reverse proxy (which, incidently, also does document caching). Also similar to the WebSEAL reverse proxy, the Edge Server Caching Proxy can be made to authenticate and authorize users against a Tivoli Access Manager security domain via the Tivoli Access Manager Plug-in for Edge Server.

The Plug-in for Edge Server incorporates an Access Manager runtime into the Caching Proxy, which allows the proxy to perform authentication and authorization based on ACLs and POPs in the Access Manager Object Space.

Because the authenticating reverse proxy functionality provided by WebSEAL is more feature-rich than that provided by the Tivoil Access Manager Plug-in for Edge Server, and because WebSEAL provides similar load balancing and content caching functions to the Edge Server, using WebSEAL is the preferred way to incorporate Tivoli Access Manager based security into the DMZ. However, the Tivoli Access Manager Plug-in for Edge Server provides an easy

way to integrate Tivoli Access Manager based security into any existing infrastructure which features an Edge Server caching proxy. In addition, the Edge Server's caching proxy provides a much more flexible set of caching options than does WebSEAL.

The following sections compare the security related aspects of the Tivoli Access Manager Plug-in for Edge Server with WebSEAL.

Access Control

In WebSEAL, as described previously, access control can be assigned at the junction level. Additionally, if more finely grained access control is needed, it can be signed to objects below the junction. The `query_contents` command is used with `pdadmin` to create these objects in the Access Manager Object Space.

When using the Plug-in for Edge Server, there is no junction object. Instead, access control is always applied directly to the objects which represent the server's content. Here also, the `query_contents` command is used, this time in conjunction with the `wesosm` command to create these objects in the Access Manager Object Space.

User Login Methods

WebSEAL provides a complete set of standard user login methods, and additonal methods are supported through customization.

The Plug-in for Edge Server is limited to Basic, Forms-based, and Certificatie-based authentication methods

Single Sign-On to WebSphere Applications

WebSeal supports Single Sign-On to WebSphere Applications via the following mechanisms:

► WebSEAL can forward modified or unmodified HTTP Basic Authentication (BA) headers to WebSphere. WebSEAL can also forward new headers based on Global Sign-on (GSO) user mapping.

► WebSphere can be made to trust the authentication performed by WebSEAL through the use of TAI or LTPA.

► When the application requires forms-based authentication, WebSEAL can submit the authentication form on behalf of the user.

In contrast:

► The Plug-in for Edge Server also supports Single Sign-On via HTTP Basic Authentication headers, however with a more limited set of filtering options. Global Sign-on usename mapping is not supported.

► The Plug-in for Edge Server supports trust relationships using LTPA cookies. TAI is not supported.

► Forms-based Single Sign-On is not supported by the Plug-in for Edge Server.

In summary, WebSEAL provides a more flexible and customizable layer of security to a WebSphere environment, when compared to the Edge Server Caching Proxy configured with the Access Manager Plug-in for Edge Server. In some cases, this may be outweighed by the more flexible caching capabiliteis of the Edge Server Caching Proxy.

For more information on the Access Manager Plug-in for Edge Server, see the following documents:

► IBM WebSphere Edge Server: New Features and Functions in Version 2, SG24-6511-00.

► Plug-in for Edge Server User's Guide, GC23-4685-00.

12.6 Scenario 2: Protecting Web resources

This scenario discovers the different techniques to protect Web resources in WebSphere using the Tivoli Access Manager.

12.6.1 Tivoli WebSEAL

WebSEAL is Access Manger's authentication engine. It is a multi-threaded Web server capable of applying security policy through Access Control Lists, ACLs, to URLs and servlets on junctioned Web servers within Access Manager's protected Web object space. WebSEAL is also where Access Manager provides Single Sign-On solutions and it is an integral part of the "defense in depth" strategy when used in its role as a reverse proxy server.

A reverse proxy server is placed in front of all other presentation layers of an application and interrupts the session flow from client Web browser to the Web server and the application servers behind them. In this role WebSEAL appears as a Web server to browser clients and appears as a Web browser to the junctioned back-end servers it is protecting.

Browser clients access what they are told by published URL, is the content server for an application. In fact this URL is a reverse proxy server. The reverse proxy is configured to hold the client session and open a new session, acting as the client with the real content server. This server will never be exposed to the real client as reverse proxy's configuration hides the contents server's name, IP address, which is also certainly a private internal and firewalled network and can remap URLs from what is published to the Web space as it actually is on the

content server. This remapping works both ways so any URLs listed in the headers being returned to the client are intercepted and rewritten to conform to the published format thus preventing external clients from getting redirection URLs to the internal content server.

In combination with Firewalls constructing DMZs to filter and direct traffic, as well as the possibility of VPNs within DMZ's there are several network designs which can strengthen the defense of an e-business Application. Decisions such as placing WebSEAL in a DMZ on its own with all the other infrastructure, Web servers, application servers and Access Manager Secure Domain servers in the second region, or placing the Web servers in the region 1 DMZ behind WebSEAL (best protected by VPNs) will be driven by both security and real production infrastructure concerns.

Figure 12-11 on page 411 shows WebSEAL functioning as a reverse proxy with the most simple model of physical network security. This server should always have dual interfaces, the first, A, is connected only to the Internet Firewall and the second, B, is connected only to the Intranet Firewall. OS level routes on the WebSEAL box should direct traffic through the intranet firewall only to those specific server that WebSEAL will need to contact, the user registry, (for authentication calls), the Policy Server (for policy database updates) and explicitly junctioned Web servers. All other traffic must be routed through interface A to the Internet Firewall.

Similar and supporting routing rules and filters must be placed on the Firewalls. The Internet Firewall must allow through traffic only to and from WebSEAL interface A and further if the site was to have only secure items then HTTPS only traffic should be also enforced. The Internet Firewall allows traffic between WebSEAL interface B and the LDAP Server, the Policy Server and the Web Server(s).
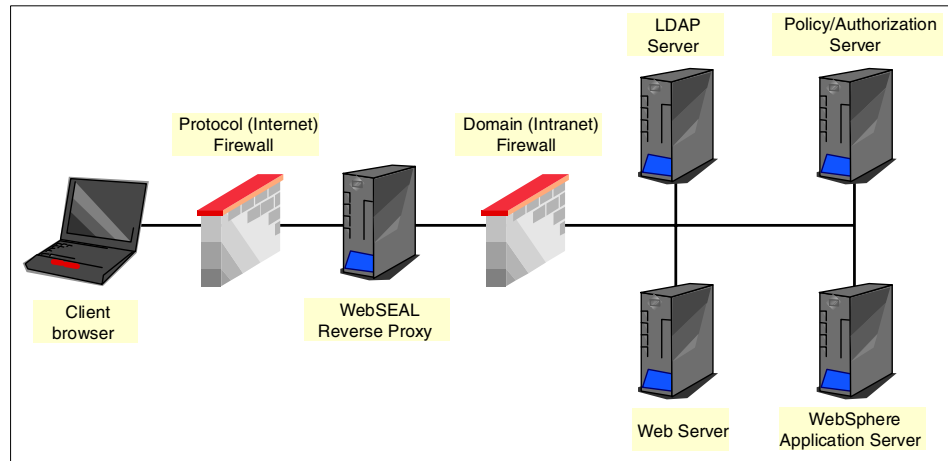
Figure 12-11   WebSEAL Basic Infrastructure and request flow

WebSEAL in the DMZ Region 1 between the Internet and Intranet firewalls receives client requests for resources of the Web server or WebSphere application server. There are five security options between WebSEAL and the Web and application servers. In each case WebSEAL authenticates users by querying the LDAP Server before connecting to any other resource.

1. WebSEAL authenticates the user, passing mapped credentials to WebSphere. WebSphere performs authorization with its own user registry.

2. WebSphere and WebSEAL, (Access Manager) use the same user registry here a common LDAP.

3. WebSEAL may also authorize the user's access to protected resources based on running a CGI program (query_contents) that accesses directory contents to determine protected files, or specific ACL lists for URLs and servlets built with pdadmin or Web Portal Manager. Authorization decisions are made from the local copy of the policy store on the WebSEAL server.

4. WebSEAL authenticates the user passing mapped credentials to WebSphere, and WebSphere-hosted applications using the Access Manager Java PDPermission or Access Manager JAAS classes which ask Access Manager for authorization.

5. WebSEAL authenticates the user passing mapped credentials to WebSphere, and WebSphere Application Server containers can delegate authorization to Access Manager through the Access Manager for WebSphere module which relies on classes in Access Manager Java Runtime and communicates with the Access Manager authorization server using the Java API. Access Manger stores role-to-user mapping only as role-to-method mapping is not yet provided.

WebSEAL Junctions

WebSEAL's connections with the back end Web servers have constantly been referred to as "junctions". This Tivoli proprietary technology requires further description in order to better understand the scenarios above.

All WebSEAL junction are TCP/IP connections between a front-end WebSEAL server and a back-end Web server which may be another WebSEAL server and may go via another proxy server. Only the HTTP and HTTPS protocols are supported and WebSEAL to WebSEAL must be over a SSL connection.

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 Sever database by apropriate use of the pdadmin command. In order to produce representations of resources on some third party back end servers within the Access Manager object space, these junctions may require configuration such that the query_contents.cgi program be loaded and accessible to be run by the Policy Server on the back end servers, themselves. This utility ships with Access Manager.

The junction is then a logical Web object space, normally 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 requested URL addresses into the physical addresses that a back-end server expects without ever 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 Policy of Protection, POP on that object which requires authentication before the request can be authorized, then the client will be challenged. WebSEAL is configurable for several different challenge mechanism including the default of Basic Authentication, forms based logon from a junctioned application and comes with an Application Developers Kit with which to build customized Cross Domain Authentication Services.

WebSEAL junctions can also be configured to enable the creation of Single Sign-On solutions allowing users to access resources, somewhat regardless of what security domain controls those resources, following their initial authentication logging on to through WebSEAL. The GSO, Global Sign On junction option allows for a third party user registry to be referenced to supply that junction with the appropriate user id and password. Other options involve manipulation and perhaps additions to the underlying Access Manger schema of

inetOrg.Person as each junction can be configured to supply all and any attributes from the schema through to the back end server. If the logon identity and passwords from the user registries of several legacy applications can be migrated into extra attributes then those applications can be accessed through WebSEAL using only one initial login. Any further login requirements from back-end servers are handled transparent to the user.

Completing the Single Sign-On picture are Cross Domain Single Sign-On and e-Community Single Sign-On. These solutions allow for the transfer of Access Manager user credentials across multiple secure domains. Please reference the Tivoli documentation at `http://www.tivoli.com/support/public/Prodman/public_manuals/td/AccessManagerfore-business3.9.html` should you wish to know more.

Creating Access Manager Groups
The ITSOBank sample application requires to create four groups and four users; Groups: managergrp, clerkgrp, accountgrp, consultantgrp. Users: manager01, clerk01, accountant01, consultant01, mdbuser01.

1. Open a browser and point it to the Web Portal Manager, https://secwpm01/pdadmin and logon as the Access Manager administrator, sec_master.
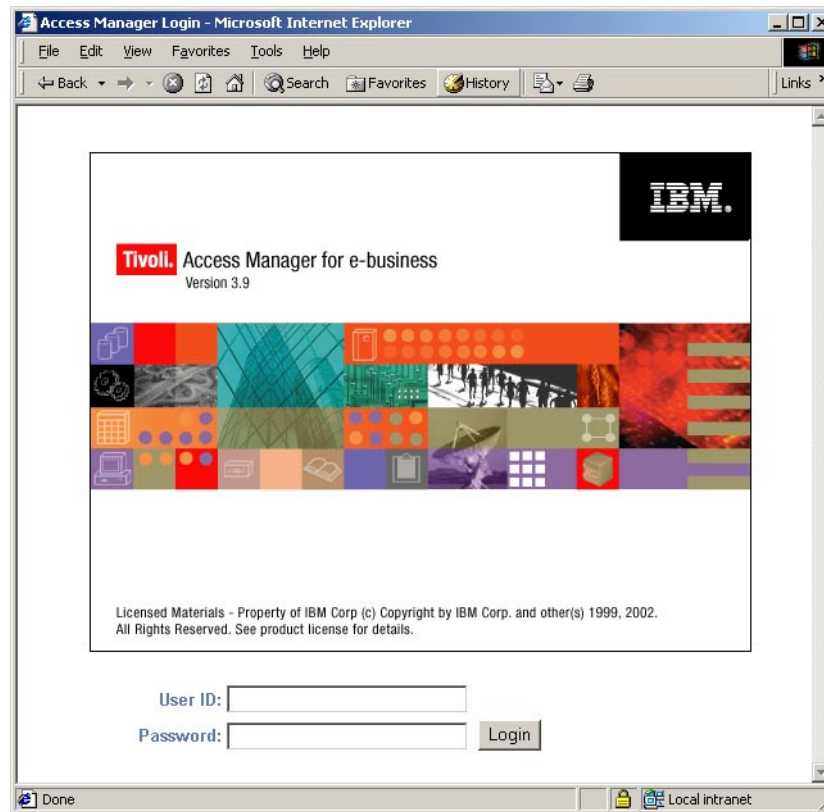
Figure 12-12   Web Portal Manager Login

Which brings up the Web Portal Manager Interface,
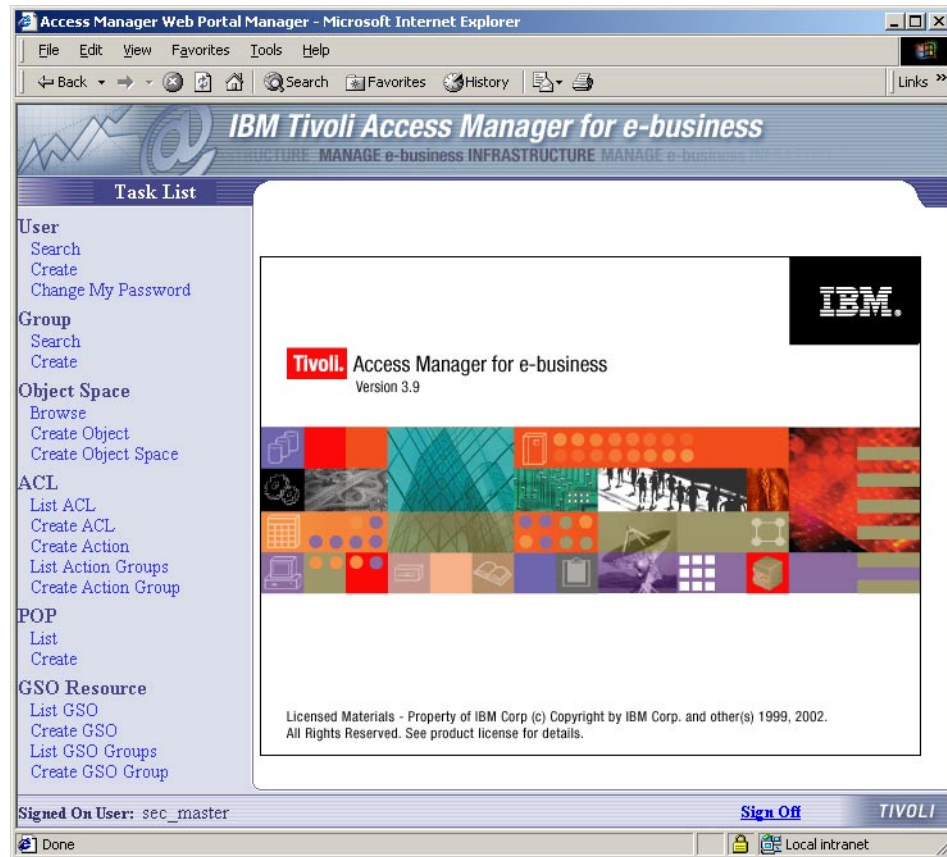
Figure 12-13   Web Portal Manager Menu

2. Click **Group -> Create** from the Task List on the left and enter these values against the named fields,

   Group Name: `managergrp`

   Description: `ITSOBank Managers`

   Common Name: `managergroup`

   Registry GID: `cn=managergrp,o=itso`

   Object Container: (can be left blank)

3. Click **Create**, a message will appear confirming group creation.

4. Repeat the process for each of the other three groups.
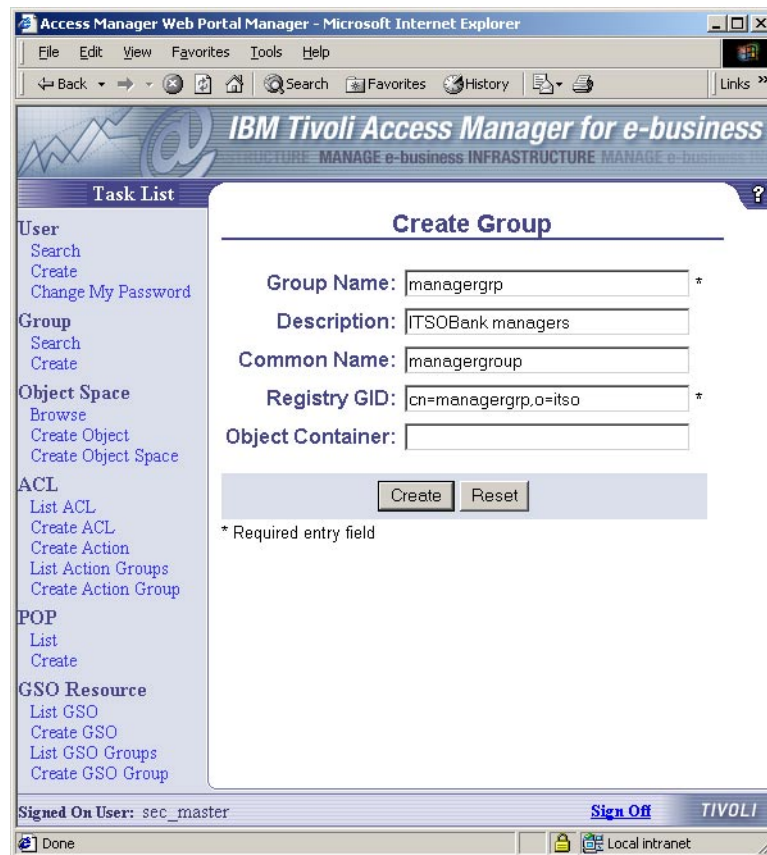
Figure 12-14   Group Creation

Creation of Access Manger Users

The following process will show, how to create a new user with the Portal Manager.

1. Click **User -> Create** from the Task List on the left and enter these values against the named fields,

   User ID: `manager01`

   For Password use `password` for this example; then confirm the password.

   Description: `Manager user`

   First Name: `Joe`

   Last Name: `Black`

   Registry GID: `cn=manager01,o=itso`

Check the boxes for **Is Account Valid**, **Is Password Valid**, **No Password Policy** and **Is GSO User**; although this last is not specifically required for these examples.

> **Note:** By checking the Is Password Valid check-box, the administrator agrees, that the password will not be checked against Tivoli Access Manager's password policy; and it reduces protection.

2. Click **List** to bring up the Group pick list window, highlight **managergrp** and click **Apply.**
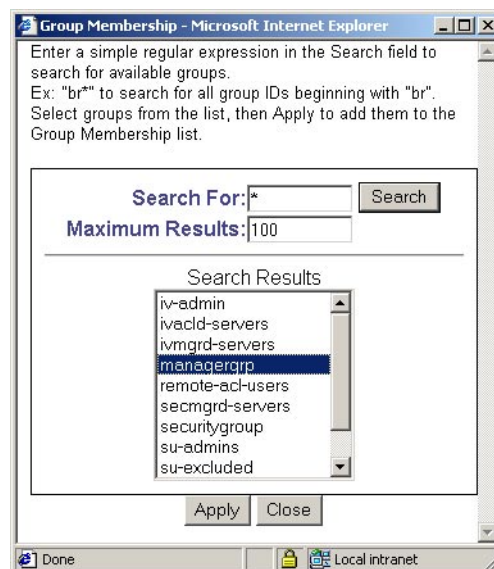
Figure 12-15   Group Pick List

Once the group is selected for the user, multiple groups can be selected, the group membership window will disappear. A banner message appears to confirm that the user has been created.

Figure 12-16   Create user

3. Click **Create** to create the user in the registry.

Create the other three users in a like manner. Creation of either users or groups can be confirmed by clicking either Search User or Search Group, both of which produce screens which will show memberships and members respectively.

The **pdadmin** command line utility could have been used to perform all group and user creation - see Access Manager Base Administrator's Guide V3.9, Appendix A for the syntax.

Creating Access Manager Junctions

At this point the configuration of the WebSphere Application Servers themselves needs some further description.

There are two separate application servers, one of which is fully configured with the secured form of the sample application with Access Manager for WebSphere installed, **appsrv01**. The setup and configuration of this instance will be gone into later in this chapter. The other has the security roles for the ITSOBank all mapped to everyone, **appsrv02,** rendering it effectively unsecured.

The two junctions needs to be created, one for each server. **pdadmin** is required to create the junctions.

1. Start the `pdadmin` command line and when the prompt appears log on as **sec_master** with the right password to the Access Manager Secure Domain using the `login` command.

2. Create a new TCP junction with the following command:

   ```
   server task WebSEALd-wsl01 create -f -t tcp -h appsrv01.itso.ral.ibm.com -p
   9080 /junction1
   ```

   This created a TCP junction (-t option) of name **junction1** on the WebSEAL server **wsl01** connected to the WebSphere Application Server **appsrv01** default host at port **9080** (-p option).

   A similar junction, **appsrv02** with `/junction2` URI is also created to the WebSphere Application Server `appsrv01`**.**

3. Log on to Web Portal Manger, select **Object Space,** then click **WebSEAL** in the tree to reveal all the objects in the WebSEAL space.Both Junctions are shown.
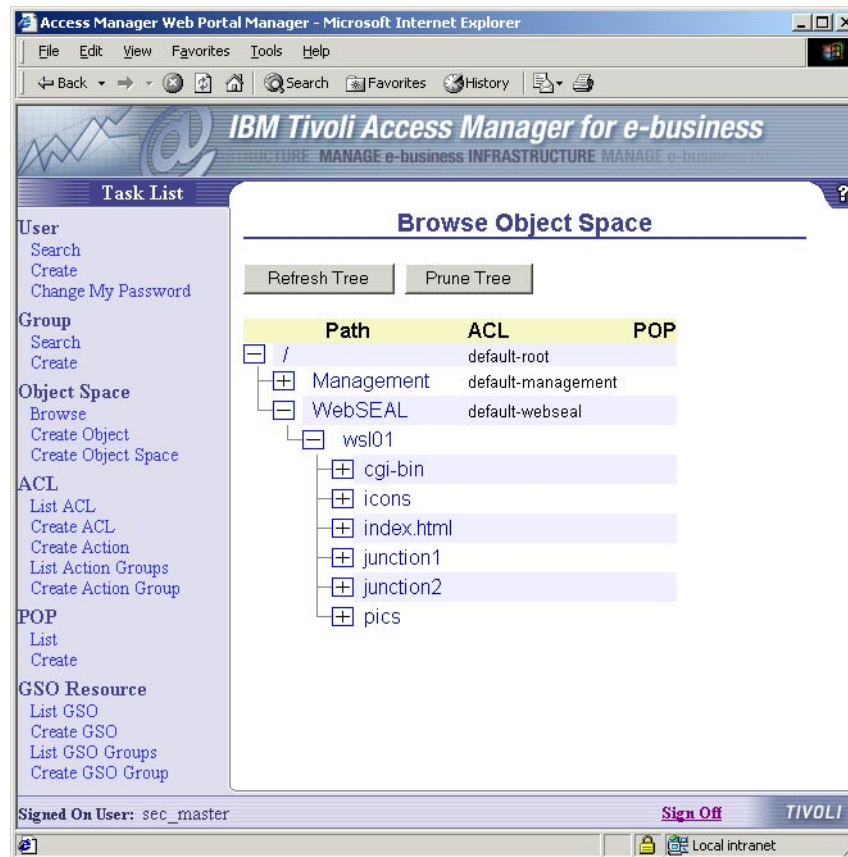
Figure 12-17   Junctions shown in Object Space Tree

4. click either one of the junctions to bring up the Protected Object Properties
   and the full name of the junction in the object space are shown, for example:
   /WebSEAL/wsl01/junction1 and **/WebSEAL/wsl01/junction2** respectively.
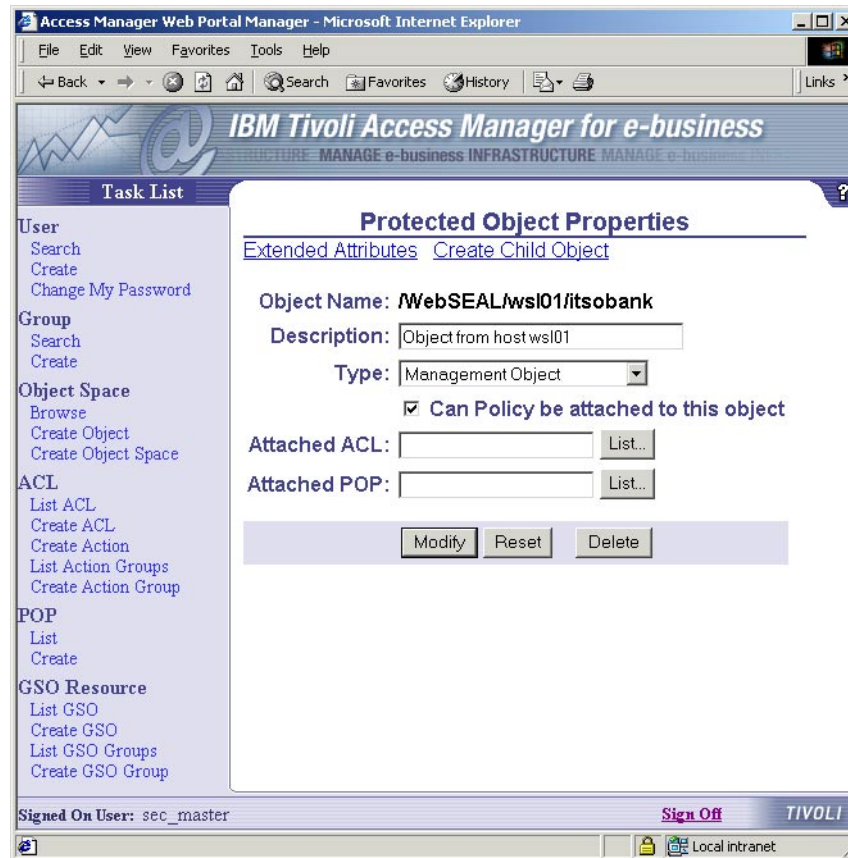
Figure 12-18   Protected Object Properties show full Junction Name

Creating ACLs

The following procedure will create ACLs (Access Control List) for the ITSOBank Web components.

1. Click **ACL -> Create ACL** from the **Task LIst** and in the **Create ACL** form which appears

2. Enter against New ACL Name: **ITSOBank**, description: **ITSOBank ACL** and click the **Create** button from the centre of the form to bring up the **ACL Properties** form. This shows one rule entry, the default permissions for sec_master. In a production environment you would very carefully consider your security model before allowing this to remain as this is exactly the kind of administrative back door discussed earlier.
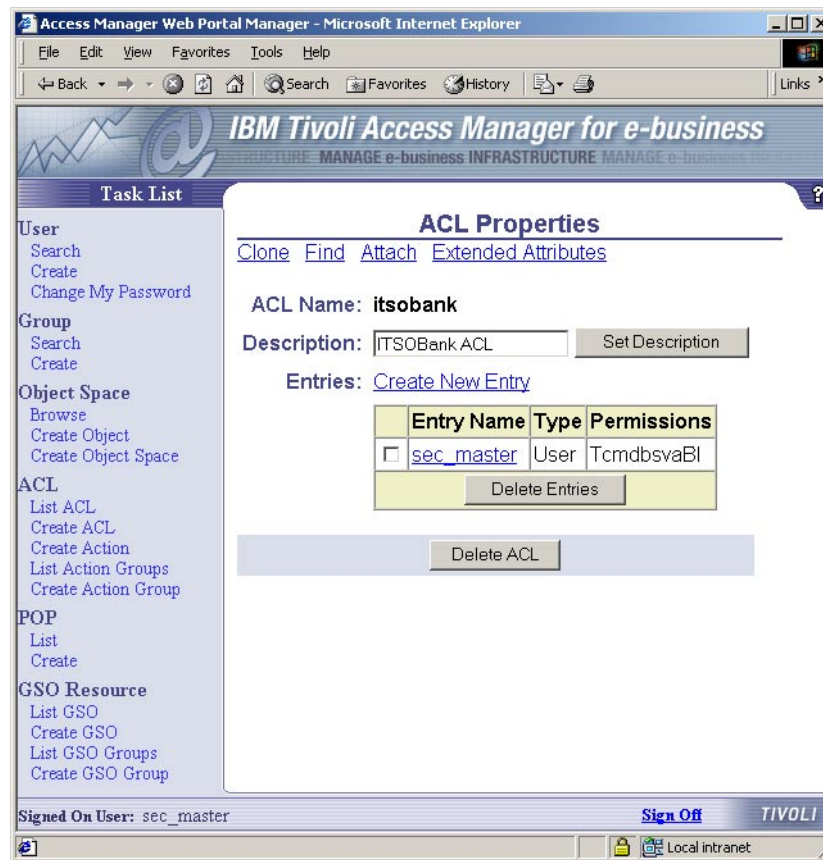
Figure 12-19   ACL Properties

3. On the page click **Create New Entry** to bring up the Create ACL Entry form, leaving the **Entry Type** at Group, enter against **Enter Name:** managergrp and check the boxes against **(T) Traverse**, **(r) Read** and **(x) Execute**, then click the button **Create Entry**.

Figure 12-20   Create an ACL Entry

4. This returns to the Create ACL form with an entry for managergrp. This
   process should be repeated for each of the other three groups.
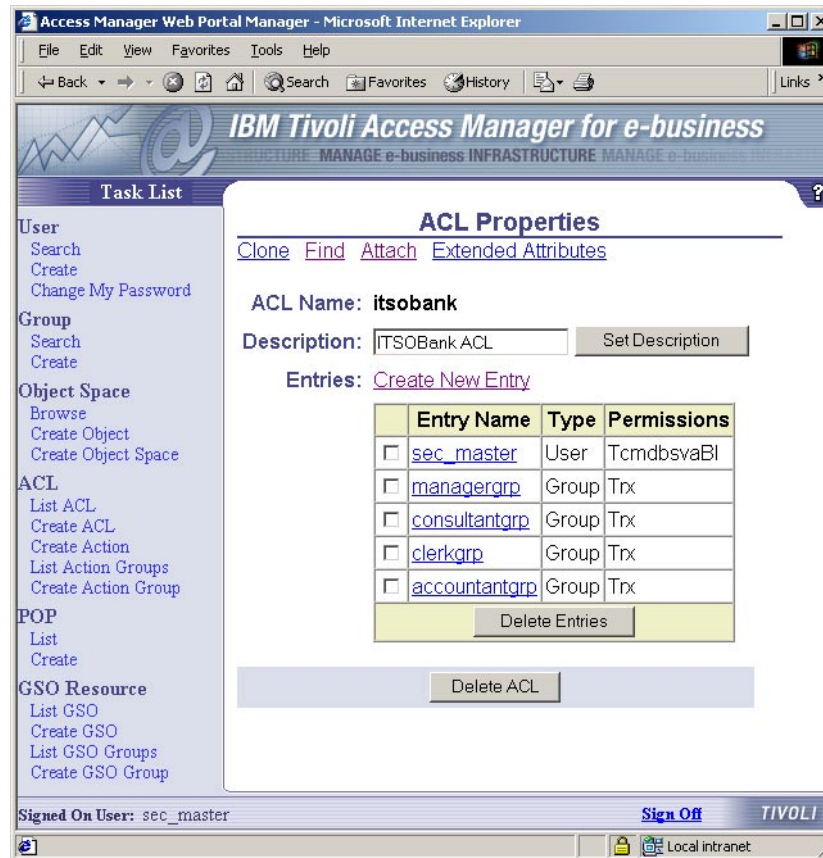
Figure 12-21   ITSOBANK ACL Properties

5. In order to attach the ACL to the appropriate junction, if not already at the ACL Properties form above, from the Task List click **ACL -> List ACL** and click the ACL **ITSOBANK** from the list to display the **ACL Properties** form.

6. Now click **Attach** to open the **Attach ACL** form. Enter the full junction name in the **Object Path:** /WebSEAL/wsl01/appsrv01 and click **Attach** which returns to the **ACL Properties** form.

7. The attachment can be checked by clicking **Find** on the ACL properties page, which opens the **ACL Find Results** form which returns all the junctions to which any ACL is attached.

8. Clicking any objects in this list will open the Protected Object Properties form which allows further attachment and detachments of ACLs and POPs to and from the object.

Testing the junctions

The following tests will help to make sure that the junctions are set correctly. The first tests will be performed on the appsrv02 server.

1. Point a browser at `http://appsrv02/itsobank` and the ITSOBank welcome screen is displayed. Click any of the functions of the application, they are available without challenge for identity.

2. Point a new browser session at `https://wsl01/junction2/itsobank/` and after the normal certificate warnings, (accept the certificate) a Basic Authentication challenge is presented which can be answered with any of the users created earlier to show the ITSOBank Welcome screen. All functionality is again available as it is only the junction which is protected.

The next two steps will access the application on the appsrv01 server.

1. Point a browser at `http://appsrv01/itsobank` and you are again presented with the welcome screen. However attempting to use either function requires a vali Access Manager identity to be authorized.

2. Point a new browser session at http://wsl01/junction1/itsobank results a Basic Authentication challenge before the ITSOBank welcome screen is shown. It is not until you attempt to access one of the protected funtions that you are challenged again for a valid identity. The welcome screen which is not protected by the application is in effective a static resource which can be independently protected by WebSEAL.

Protecting WebSphere URIs

Access Manger can secure WebSphere servlets and JSPs but not EJBs or individual methods, only URIs. If the application you wish to secure is of a simple enough design then this may be all you need.

This sample will use the index.jsp page provided with the ITSOBank sample application. The index.jsp is available for everyone, the resource is not protected, access is granted for the Everyone special subject to be exact. This sample will use Tivoli Access Manager WebSEAL to protect this resource and only give access to the user `accountant01` in the `accountantgrp` group. You can try to access the index.jsp at `http://<your_server>/itsobank/index.jsp`.

1. Start the pdadmin tool to administer Tivoli Access Manager, and login with the **sec_master** user.

2. Create a junction to the Web space.

   ```
   pdadmin> server task Webseald-wsl01 create -f -t tcp -h appsrv01 -p 9080
   /itsobankURItest
   ```

3. Create a new ACL for the index.jsp resource.

   ```
   acl create itsobankURItestACL
   ```

4. Modify the ACL to add the accountantgrp user group with the Trx permissions.

```
acl modify itsobankURItestACL set group accountantgrp
```

5. Attach the ACL to the index.jsp resource in the object space.

```
acl attach /WebSEAL/wsl01/itsobankURItest/itsobank/index.jsp
itsobankURItestACL
```

6. Double check the ACL settings and attachment using the **acl show itsobankURItestACL** command then the **acl find itsobankURItestACL**. The result should be similar to the following:

```
pdadmin> acl show itsobankURItestACL
    ACL Name: itsobankURItestACL
    Description:
    Entries:
        User sec_master TcmdbsvaBl
        Group accountantgrp Trx
pdadmin> acl find itsobankURItestACL
/WebSEAL/wsl01/itsobankURItest/itsobank/index.jsp
```

To test the protected resource access the index.jsp through the WebSEAL server using the following URL pattern:
https://wsl01/itsobankURItest/itsobank/index.jsp. After a confirmation of accepting the server certificate, you will be presented with the browser's basic authentication challenge panel. Use `accountant01` / `password` for user name/password to access the page. The page you get is shown on the next screen capture.
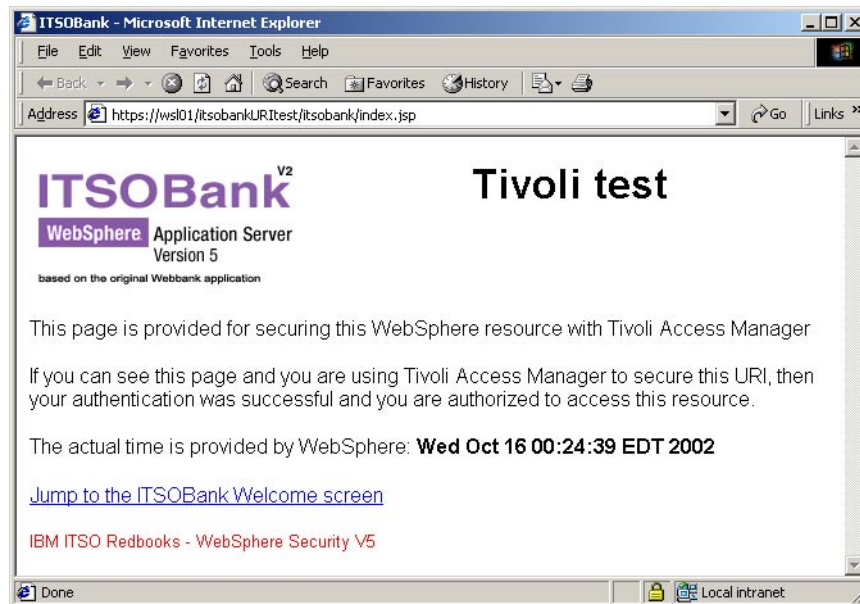
Figure 12-22   Successful access to index.jsp

Close the browser and access the same URL again, but use another user when prompted, for example: `manager01` with `password`. You should get the following page.
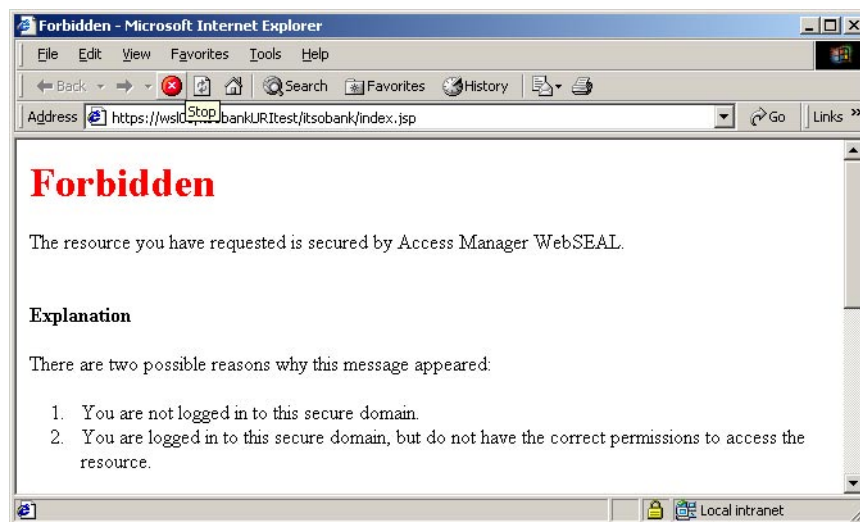
Figure 12-23   Unsuccessful access to index.jsp

It is possible to set up a hierachy of access based on the representation of the Web Server's resources in the Tivoli Access Manager Object Space. Rights and permissions cascade within the space and so each higher level will have increasing more generic protection but it is mandatory that any group or user given access at the lowest level, for example the resource must also be presented in an entry within any other ACLs higher in the object space. In this example if the entry for accountantgrp is removed form the ITSOBANK attached to the object /WebSEAL/wsl01/itsobankURItest, from the previous sample, then any attempt by accountant01 to access a resources deeper in the object space, here /WebSEAL/wsl01/itsobankURItest/itsobank/index.jsp despite the itsobankURItestACL attached to this object, will fail as the Transverse right for accountantgrp group is discontinuous at the higher level.

12.7 Scenario 3: Tivoli's WebSphere plug-in

This scenario documents, how to use the WebSphere plug-in from Tivoli Access Manager to control WebSphere security from the Tivoli Access Manager.

12.7.1 Access Manager For WebSphere Application Server

An extension of Access Manager Version 3.9 provides container-based authorization and centralized policy management for IBM WebSphere Application Server applications. Effectively Access Manager provides a J2EE Authorization Module which when installed correctly replaces WebSphere's own security for user role based authorization decisions.
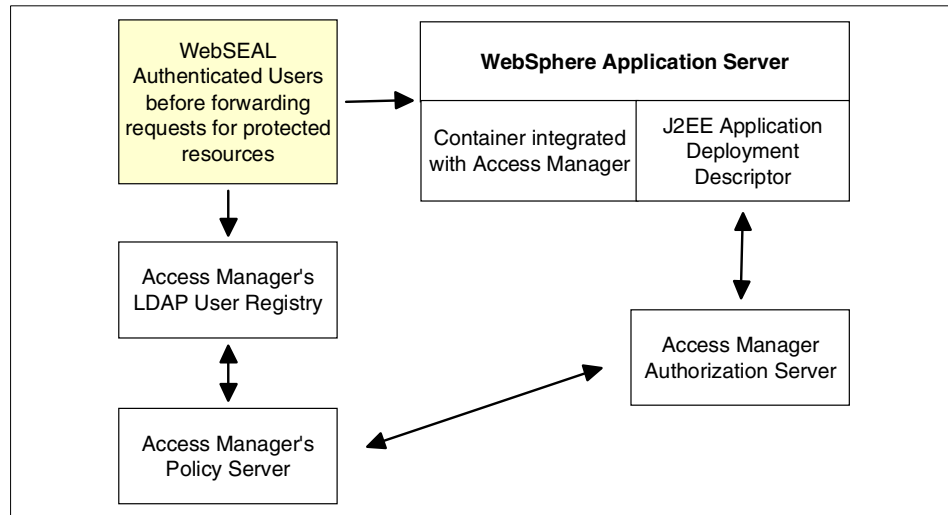
Figure 12-24   Access Manager for WebSphere beta Model

When a user requests a protected resource, WebSEAL authenticates the user against the Access Manager user registry. Junction configuration defines the type and number of credentials then forwarded to the application server.

The container examines the request for access to a protected resource and from the J2EE application deployment descriptor, determines the required role that the user must have to be granted authorization. The container then hands off to the integrated Access Manager module.

The Access Manager module requests an authorization decision from an Access Manager authorization server which checks with its local replica of the Access Manager policy database. Replicas are normally updated on a pull basis from the single Access Manager Policy Master within the Security Domain.

While these calls can be made to a remote server, without the embedded Access Manager promised for the final WebSphere Application Server V5, performance and scalability require that an Access Manager Authorization server be installed on the same platform as WebSphere. Of course this means that the performance burden is passed to the platform hardware which must be capable of bearing both loads.

Having returned the access decision, granted or denied, to the container, WebSphere then acts on it.

Without an Access Manager authentication blade such as WebSEAL, a Web server with an Access Manager plug-in or WebSphere Edge Server with the Access Manager plug-in, the WebSphere container would request authentication of the user from the user registry directly. The authorization process would however then be identical.

The advantages of externalizing the control of the security model have been discussed earlier. The specific advantage of the integrated Access Manager module is that J2EE applications do not require coding changes to take advantage of the dynamic flexibility allowing changes to user and group mappings to roles without stopping and starting an application through the manipulation of Access Manager ACLs.

The highest prefix to all J2EE roles defined for WebSphere applications is the Access Manager protected object for "WebAppServer" together with the child object "deployedResources". Both these object names are created the first time the Access Manager application migration tool is run.
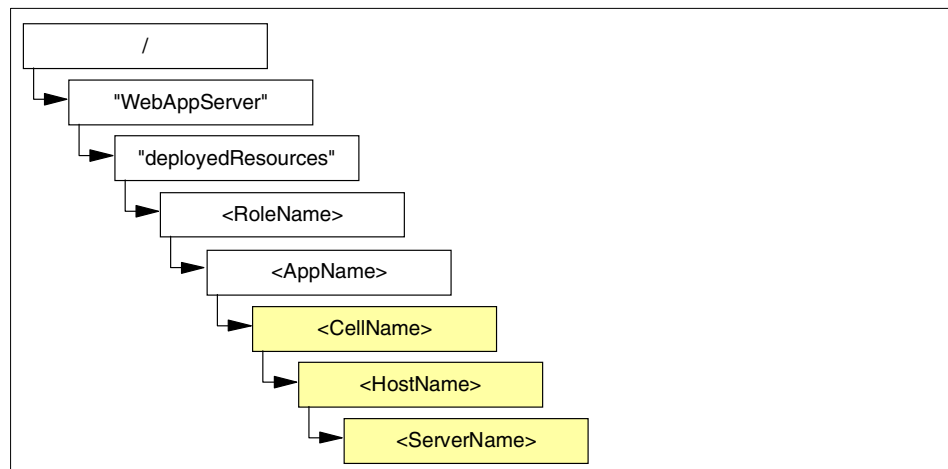
Figure 12-25   Access Manager Protected Object Name Space

Figure 12-25 shows the Access Manager name space to which users and groups are associated with roles and applications and optionally, down to the cell, host or server. ACLs can be placed at any point in this name space determining which J2EE roles any principal governed by those ACLs has been assigned. The Access Manager migration tool, automatically attaches the ACLs at the "AppName" level. As with all ACLs in the object space, the permissions must be continuos and the lower level ACL over rides a higher ACL.

At the same time, the tool creates a "WebAppServer" action group with the invoke (i) action, and a group called "pdwas-admin" representing WebSphere Application Server administrators. The tool adds the WebSphere Application Server administrator user to the pdwas-admin group.

Installation and configuration of Tivoli Access Manager for WebSphere

Install the Access Manager for WebSphere application on the WebSphere machine. There is no ez_install script provided for this component, you have to install it from the product CD. The Access Manager for WebSphere can be found in the following directory: windows\policy director\disk images\disk1\pdwas\disk images\disk1. Run the setup.exe to install the product. In this sample Tivoli Access Manager components were installed under the C:\Tivoli direcotry.

> **Note:** At the time of writing this book Tivoli Access Manager V3.9 was only available. This version of Access Manager was not developed to be used with WebSphere Application Server V5, therefore we had to do some customization and work arounds to make certain scripts and functions work in this environment.
>
> Tivoli Access Manager V3.9 is available after the book is published and it works with and supports WebSphere Application Server V5.

The following configuration steps are required in WebSphere Application server in order to perform further configurations and use Access Manager for WebSphere.

1. The WebSphere Application Server must be configured to share the same user registry as the Access Manager Security Domain it is joining.

2. The Access Manager for WebSphere module must be installed and configured.

3. J2EE Applications requiring security must be migrated.

4. All user ids which had been used with WebSphere, wasadmin (the server id), and the other Access Manager users and groups required by the sample application had been created with Web Portal Manager so no migration of WebSphere only LDAP users was required.

5. Confirm that Access Manger and WebSphere were accessing the same Java runtime was confirmed by running the *pdjretecfg* utility, under Windows in a command prompt.

```
cd C:\Tivoli\sbin
pdjrtecfg -action config -java_home %WAS_HOME%\java\jre.
```

6. The next step is to run the **PDWASCFG** utility for which there are a number of inputs.

   A user account to be the user identity for the Access Manager for WebSphere application: wasadmin.

   The sec_master password and the fully qualified name of the Access Manager server- password and secsrv01.itso.ral.ibm.com respectively.

   And the name of the Authorization Server that WebSphere would be accessing, here the same server WebSphere was running on, appsrv02.itso.ral.ibm.com.

   Opening a Windows command prompt window the following commands were executed:

```
cd C:\Tivoli\sbin
set PDWAS_HOME=C:\Tivoli\pdwas
set WAS_HOME=C:\WebSphere\AppServer
set
CLASSPATH="%PDWAS_HOME%\sbin";"%PDWAS_HOME%\lib\PDWASAuthzManager.jar";"%PD
WAS_HOME%\lib";"%CLASSPATH%"
java -Dpdwas.home="%PDWAS_HOME%" -Dwas.home=%WAS_HOME% -cp %CLASSPATH%
PDWAScfg -action config -remote_acl_user "cn=wasadmin,o=itso"
-sec_master_pwd password -pdmgrd_host secsrv01.itso.ral.ibm.com
-pdacld_host appsrv02.itso.ral.ibm.com
```

   The success of the action was confirmed by checking the existence of the PdPerm Properties file, c:\WebSphere\Appserver\java\jre\PdPerm.properties

   Configure the Access Manager authorization component for WebSphere Application Server. The meanings of the parameters for the PDWAScfg utility:

   – **action** specifies the command to perform. It is either configuration or unconfiguration. The valid values for this option are config or unconfig.

   – **remote_acl_user** is the full DN of the remote acl user, used for the SSL connection with the Access Manager authorization server.

   – **sec_master_pwd** is the password of the sec_master user.

   – **pdmgrd_host** contains the hostname of the Access Manager policy server.

   – **pdacld_host** contains the hostname of the Access Manager authorization server.

   – **pdmgrd_port** the port number of the Access Manager policy server can be specified if it has been configured as different from the standard port.

   – **pdacld_port** the port number of the Access Manager authorization server can be specified if it has been configured as different from the standard port. Note that pdmgrd_port must also be specified if this option is used.

– **rspfile** is the fully qualified name of the response file to use during silent installation. This is an optional option.

For more information about the parameters and using the PDWAScfg utility refer to the original product documentation.

12.7.2 Migration of Applications

In order for the Access Manager module to perform the authorization functions for J2EE applications deployment descriptors security information needs to be migrated into Access Manager's object space. The *migrateear.jar* tool which ships with Access Manager functions as did the *PDWAScfg* tool - a standalone Java application using Access Manager Java Admin APIs. It creates a tree of roles, see Figure 12-25 on page 430 and ACLs of users and groups and maps the invoke permission from the WebAppServer object to these ACLs.

> **Important:**
>
> Because Tivoli Access Manager V3.9 is not fully prepared to work with WebSphere Application Server V5, there are some additional steps and tweaking required to make it work.
>
> The version of **xerces.jar** shipped with WebSphere Application Server Version 4 was copied to the **%PDWAS_HOME%\lib** directory.
>
> The **appliaction_1_2.dtd** and **application_1_3.dtd** were copied from **%WAS5_HOME%\deploytool\itp\plugins\com.ibm.etools.j2ee\dtds** to the **%PDWAS_HOME%\etc** directory.

1. Start the pdadmin administration application.

2. Logon to pdadmin as sec_master and create an Access Manger action and action group as follows.

   ```
   action group create WebAppServer
   action create i invoke "Invoke WebAppServer"
   ```

3. Then exit from pdadmin but remain in the windows command prompt and change directory to **%PDWAS_HOME%/bin**.

> **Note:** When performing the following step WebSphere Application Server should not be running.

```
set PDWAS_HOME=C:\Tivoli\pdwas
set WAS_HOME=C:\WebSphere\AppServer
set XML_PARSER_JAR=%PDWAS_HOME%\lib\xerces.jar
```

```
set JDK_DIR=%WAS_HOME%\java\jre
set PDWAS_JAR=%PDWAS_HOME%\lib\migrate.jar
set CLASSPATH="%XML_PARSER_JAR%";"%PDWAS_JAR%";"%CLASSPATH%"
```

4. The first appliaction to be migrated is the adminconsole itself.

```
java -Dpdwas.lang.home=%WAS_HOME%\lib;%PDWAS_HOME%\nls\java -cp %CLASSPATH%
com.tivoli.pdwas.migrate.Migrate -j
%WAS_HOME%\installedApps\appsrv01Node\adminconsole.ear -a sec_master -p
password -w wasadmin -d o=itso -c
file:/%WAS_HOME%/java/jre/PdPerm.properties
```

The migration utilty created the four roles as defined in the Admin Console application and the AppName, Admin Console attached an automatic ACL to this level. See details on how to check the created objects later in this section.

5. As a next step the console role ACLs need to be attached to relevant groups in Access Manager. The following example attaches the pdwas-admin group to the *_WebAppServer_deployedResources_administrator_Admin_20_Console_ACL* ACL. Login to the pdadmin administration utility as sec_master, then perform the following commands:

```
pdadmin> acl modify
_WebAppServer_deployedResources_administrator_Admin_20_Conso
le_ACL set group pdwas-admin Ti
pdadmin> acl show
_WebAppServer_deployedResources_administrator_Admin_20_Console
_ACL
   ACL Name:
_WebAppServer_deployedResources_administrator_Admin_20_Console_ACL
   Description: Generated by the PDWAS Migration Tool
   Entries:
       User sec_master TcmdbsvaBl
       Group pdwas-admin Ti
```

6. Do the same with the other console ACLs:

```
_WebAppServer_deployedResources_monitor_Admin_20_Console_ACL
_WebAppServer_deployedResources_operator_Admin_20_Console_ACL
_WebAppServer_deployedResources_configurator_Admin_20_Console_ACL
_WebAppServer_deployedResources_administrator_Admin_20_Console_ACL
```

7. Restart the WebSphere Server before migrating any other aspplications.

8. The next appliaction migrated was the ITSOBank sample application.

```
java com.tivoli.pdwas.migrate.Migrate -j
\WebSphere\AppServer\installedApps\appsrv01Node\ITSOBank.ear -a sec_master
-p password -w wasadmin -d o=itso -c
file:/%WAS_HOME%/java/jre/PdPerm.properties
```

The meanings of the parameters for the migration utility:

- **-a** <Access Manager administrative user

  For example: `-a sec_master`.
- **-c** <URI location of the PDperm.properties>
- **-d** <user registry domain suffix>
- **-j <**absolute pathname to the application EAR file>
- **-p** <administrative user password>
- **-r** <root object space and action group name>

  The default value for the root object space is WebAppServer. The action group name matches the root object space name. Thus, the action group name is automatically set when the root.
- **-t** <Secure Sockets Layer timeout>

  The maximum should not exceed the Access Manager SSL-v3-timeout value. The default value for SSL-v3-timeout is 120 minutes.
- **-w** <WebSphere administrative user>

9. After migrating the applications check the objects created for Access Manager. Issue the following commands to see the objects in the objectspace:

```
pdadmin> login
Enter User ID: sec_master
Enter Password:

pdadmin> objectspace list
/Management
/WebSEAL
/WebAppServer/deployedResources

pdadmin> object list /WebAppServer/deployedResources
    /WebAppServer/deployedResources/accountant
    /WebAppServer/deployedResources/administrator
    /WebAppServer/deployedResources/allauthenticated
    /WebAppServer/deployedResources/clerk
    /WebAppServer/deployedResources/configurator
    /WebAppServer/deployedResources/consultant
    /WebAppServer/deployedResources/manager
    /WebAppServer/deployedResources/monitor
    /WebAppServer/deployedResources/operator
```

List the ACLs in Access Manager and check if you have all the console role ACLs and ACLs for each J2EE role in your application(s).

```
pdadmin> acl list
default-webseal
default-root
```

```
_WebAppServer_deployedResources_monitor_Admin_20_Console_ACL
_WebAppServer_deployedResources_operator_Admin_20_Console_ACL
default-gso
itsobank
itsobankURItestACL
_WebAppServer_deployedResources_consultant_ACL
_WebAppServer_deployedResources_configurator_Admin_20_Console_ACL
default-policy
_WebAppServer_deployedResources_accountant_ACL
_WebAppServer_deployedResources_clerk_ACL
default-config
_WebAppServer_deployedResources_manager_ACL
default-management
_WebAppServer_deployedResources_administrator_Admin_20_Console_ACL
_WebAppServer_deployedResources_allauthenticated_ACL
default-replica
```

You can check the details on one of the ACLs using the following command:

```
pdadmin> acl show
_WebAppServer_deployedResources_monitor_Admin_20_Console_ACL
    ACL Name: _WebAppServer_deployedResources_monitor_Admin_20_Console_ACL
    Description: Generated by the PDWAS Migration Tool
    Entries:
        User sec_master TcmdbsvaBl
        Group pdwas-admin Ti
```

For further detials, check the objects which the ACL is attached to.

```
pdadmin> acl find
_WebAppServer_deployedResources_monitor_Admin_20_Console_ACL
/WebAppServer/deployedResources/monitor/Admin Console
```

For more information about the parameters and using the migration utility refer to the original product documentation.

After Migration

Once an appliaction is migrated, its security is in the province of the enterprise security model and should be controlled using Access Manager, either Web Portal Manager or the pdadmin utility. This is especialy true for the modification of existing or the creation of new roles.

Do not make changes the deployment descriptors of an application from within WebSphere. They are not reflected in the EAR file and can not be captured by Access Manager. Thus it behooves you to ensure that if you are migarting existing appliactions designed before the enterprise security model was in place that the EAR file you migrate accurately reflects the application's current security configuration.

The migration of an EAR file to the Access Manager protected object space creates ACLs attached to those objects. If these ACLs are used elsewhere within the object space they cannot be removed while attached to any object.

12.8 Scenario 4: Using the aznAPI

The IBM Tivoli Access Manager Java runtime component includes a Java version of a subset of the Access Manager authorization API. 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.

Application developers should use the Javadoc information provided with the Access Manager Application Developer Kit (ADK) to add Access Manager authorization and security services to new or existing Java applications.

The authorization API classes are installed as part of the Access Manager Java runtime component. These classes communicate directly to the with the Access Manager authorization server by estabilishing an authenticated SSL session with the authorization server process.

> **Important:** The aznAPI install as the Application Development Kit of Tivoli Access Manager. When you use the ezinstall for Access Manager to install the product it will not install the authorization API for you.
>
> In order to get the API installed on your system, you have to install it manually from the Tivoli Access Manager CD. You can find the install image called: PDJRTE, which installs the Java Runtime component for Tivoli Access Manager. On the Windows platform the PDJRTE is at: `\windows\PolicyDirector\Disk Images\Disk1\PDJRTE`.

The aznAPI Java classes are basically Java wrappers for the original C API.

The aznAPI can be used together with Java 2 security; where Java 2 security consists of the policy-based permission model and the JAAS extension for authentication. Access Manager functions as a back-end for normal Java 2 permission checks by providing:

▶ A custom JAAS LoginModule that manufactures authentication credentials.

▶ A custom permission class that knows how to locate and call Access Manager.

For more information refer to the Authorization Java Classes Developer's Reference documentation that comes with the rest of the Tivoli Access Manager 3.9 documentation.

Part 3

Appendixes

439

A
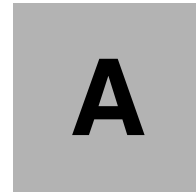
Sample application

This appendix introduces the ITSOBank sample application used in this book to show the security settings in WebSphere Application Server V5.

Here you will find detailed information about the application itself, the application design and building blocks.

The most important is the installation procedure that guides you through the installation steps for the sample application.

Sample application

The pupose of the sample application is to show the security functions and capabilities of WebSphere Application Server V5. You will not find this application in a real-life scenario, it has been developed only for this book and is not a realistic implementation of any banking application.

The sample that you can download toghether with the book has all the security features built in, that was discussed in the book. All security settings are fully configured for the application too.

Application architecture brief

This section provides a brief introduction for the sample application used in this book. The source code is also provided with the code, and it is well commented. You will find it easy to understand every components and parts.

The following diagram is a collection of the application resources used by the ITSOBank sample.
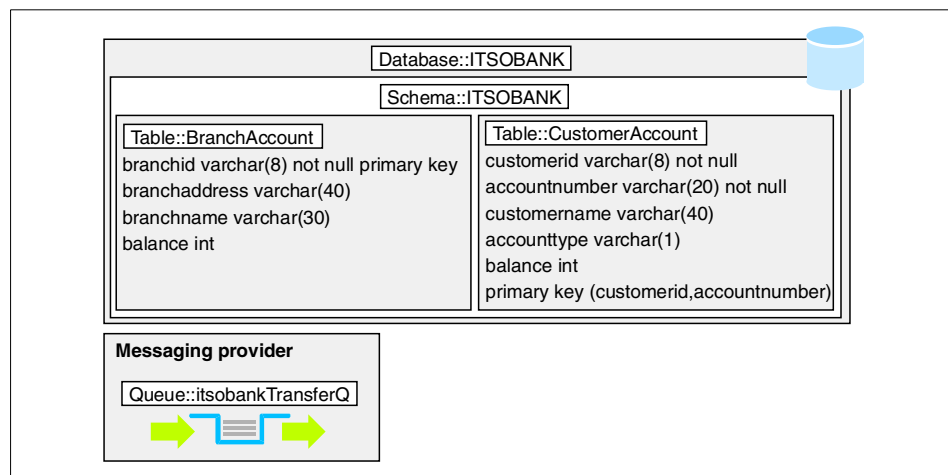
Figure 12-26   Application Resources

Instead of providing component and class diagrams, the application is so simple that you will understand it better by going through a couple technical walkthroughs.

The first walkthrough descibes the "customer transfer" process, as it is depicted on Figure 12-27.
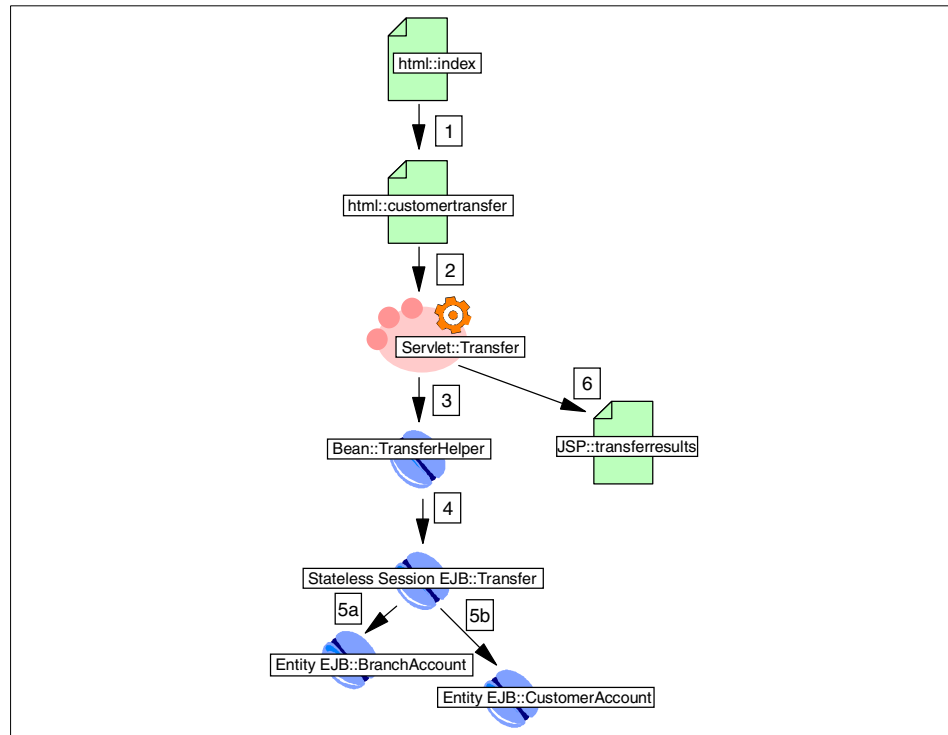
Figure 12-27   Customer transfer process

The process flows as described below:

1. User selects the "customer transfer" link on the main page (index.html) of the application.

2. After filling out the details on the transfer form, the user submits the transfer to the Transfer servlet.

3. The transfer servlet is the controller between the presentation and the business logic represented by enterprise Java beans (EJB). The servlet processes the request. You will find multiple functions and features implemented in this simple servlet, these functions will be executed according to the request. You can find more information about these functions in the book where specific security features are discussed and you will find a reference to the source in the code.

4. The servlet uses a singleton helper class (TransferHelper) to lookup and store the reference to the Transfer session EJB. The Transfer EJB is a stateless session EJB, it implements the business processes and handles the transfer between the accounts. It follows the facade pattern, and hides the transfer logic between multiple entity EJBs.

5. The Transfer EJB uses two entity EJBs: CustomerAccount and BranchAccount to make the transfer between accounts and update the back-end system, which is a database in our case.

6. After the transfer the Transfer servlet sends the response back to the client using a JSP.

The next process depicted on Figure 12-28 is a transfer between two branches, and JMS messaging is used to process the request.

The first four steps are the same as with the "customer transfer".
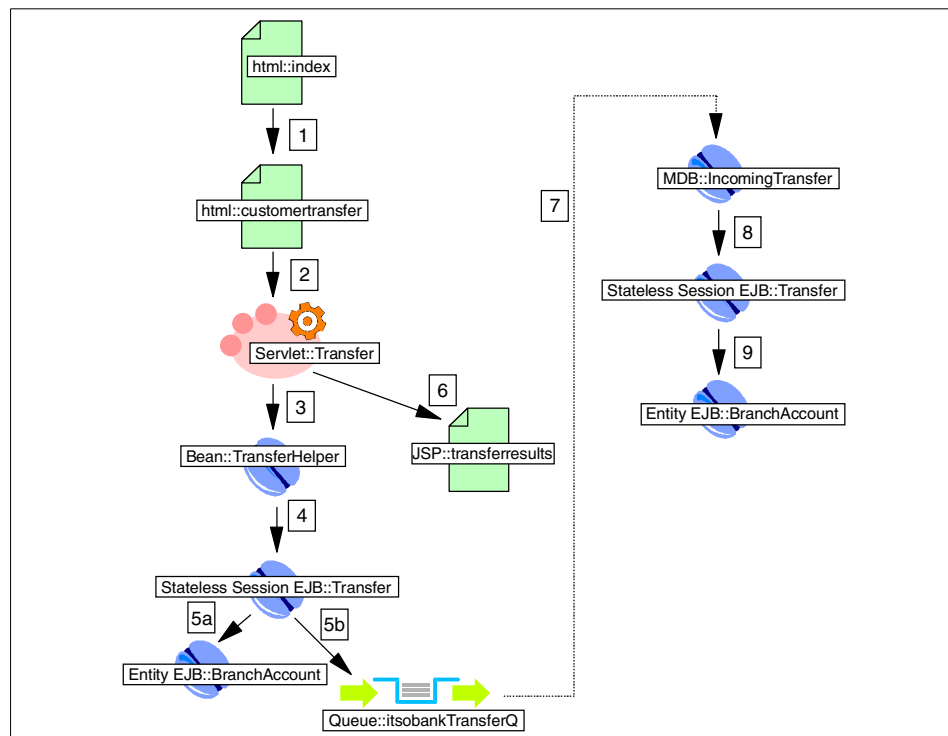
Figure 12-28   Branch transfer process

The process flows as described below:

1. User selects the "customer transfer" link on the main page (index.html) of the application.

2. After filling out the details on the transfer form, the user submits the transfer to the Transfer servlet.

3. The transfer servlet is the controller between the presentation and the business logic represented by enterprise Java beans (EJB). The servlet processes the request.

4. The servlet uses a singleton helper class (TransferHelper) to lookup and store the reference to the Transfer session EJB.

5. The Transfer EJB updates the branch account at the back-end using the BranchAccount entity EJB, then sends a message to the queue, itsobankTransferQ, to update the other branch account.

6. Same as "customer transfer".

7. The message triggers the IncomingTransfer message-driven bean (MDB) that picks up the message to process the account update for the branch account.

8. The MDB does not perform business logic, it uses the Transfer EJB for this purpose.

9. The Transfer EJB uses the BranchAccount entity EJB to update the back-end database.

The last process described here is the account balance query. A J2EE client application is also distributed with the sample, that performs the query process. It is a simple GUI application that the user can use to interact with the J2EE application running on the server.
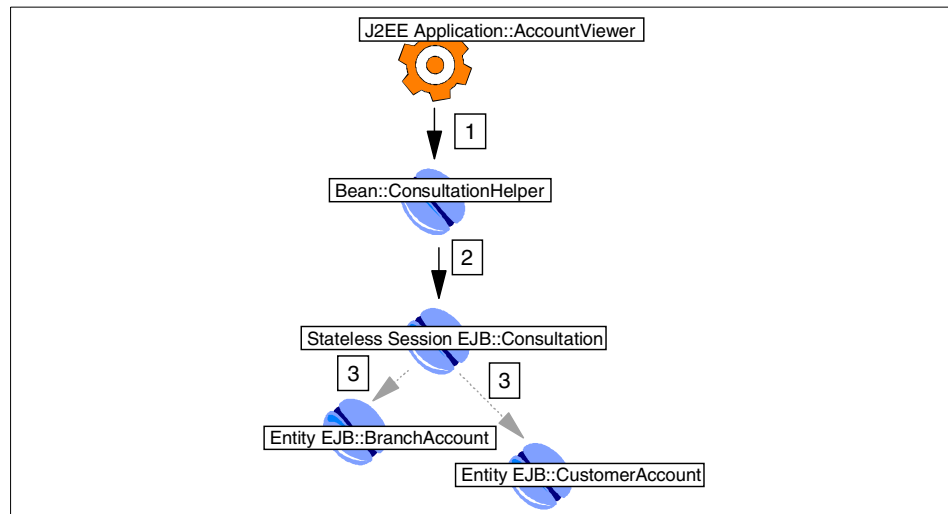
Figure 12-29   Account balance query process

The process flows as described below:

1. The user provides the account name for the customer or the branch then clicks the query button. The application uses a singleton helper class (ConsultationHelper) to lookup and store the reference to the Consultation session EJB.

2. The Consultation EJB is a stateless session EJB, it performs the query on the server side. IT follows the facade pattern to hide the logic at the back-end.

3. The Consultation EJB performs a query on one of the accounts using either the BranchAccount or CustomerAccount entity EJBs, and returns the result to the application.

To follow the processes in the runtime, you can check the log file for the application server, SystemOut.log. The detailed runtime information about the application can only be found there, for example: the branch account update using JMS messaging, or detailed information about the client's subject.

Security roles

The following roles are defined for the secured ITSOBank sample:

► manager

► clerk

► accountant

► consultant

► mdbuser

► AllAuthenticated

► Everyone

> **Note:** MDBUser is created to be used for delegation. The IncomingTransfer MDB is using this role to call the Transfer Session Bean.
>
> There should be one user mapped to this role; that user will be used to do the run-as mapping.

Deploying the sample application

This section provides step-by-step instructions on how to deploy the ITSOBank applications.

Setup the database server

The ITSOBank sample, like any other enterprise application, requires a database.

1. Create a user in the operating system with the username: **dbuser** and password: **password**. this user will perform the database access for the sample application.

2. Install the DB2 UDB Server on the database server machine.

3. Stop all the DB2 services and daemons, you can use the command: *db2stop*.

4. Open a console, go to the DB2 UDB Client install directory, then go to the java12 directory there. Execute the *usejdbc2.bat* (on Windows) or the *usejdbc2.sh* (on UNIX) to switch to JDBC 2.

5. Start the database server, you can use the command: *db2start*.

6. Bind the necessary packages it is required for JTA drivers, from a DB2 console, you can use the following commands to bind the packages on the server.

```
connect to itsobank user dbuser using password
bind '<db2 install path>\bnd\@db2cli.lst'
bind '<db2 install path>\bnd\@db2ubind.lst'
disconnect itsobank
```

Where *<db2 install path>* is the install path for DB2, replace with the correct path.

7. Open a console for DB2 command execution, and run the following commands:

```
db2 attach to db2 user db2admin using db2admin
db2 create database itsobank
db2 create schema itsobank
db2 connect to itsobank
db2 grant dbadm on database to user dbuser
db2 disconnect current
db2 connect to itsobank user dbuser using password
db2 -vf itsobank.sql
```

Note: in the previous commands the following assumptions were made:

► The database instance is called: *DB2*

► There is a user called *dbuser* with the password *password*, this user will bu used for the database connection in the datasource.

In case you have any problems setting up the server, refer to the official product documentation of IBM DB2 UDB.

Setup the database client

In case the application server is running separate from the database server the database client has to be installed on the application server's machine. It is optional to have the application server separate, however it is recommended.

> **Note:** Where you can download the DB2 client:
>
> `ftp://ftp.software.ibm.com/ps/products/db2/fixes/english-us`

You can download the clients from here. First you have to select the directory for the platform and versoin you are looking for; for example: db2aixv7 (AIX Version 7.x). Go down to the client then to the runtime sub-directory. There you will find a bunch of files, one for each fix-pack, download the one you need; for example: `FP7_U482564_rtclnt.tar.Z` (fixpack 7). For AIX, DB2 Version 7, fixpack 7, the full URL for the file is:

`ftp://ftp.software.ibm.com:21/ps/products/db2/fixes/english-us/db2sunv7/client/`
`runtime/FP7_U482564_rtclnt.tar.Z`

Instructions to setup the database client for the sample application.

1. Install the DB2 Database Client on the machine.

2. Open a console, go to the DB2 UDB Client install directory, then go to the java12 directory there. Execute the usejdbc2.bat (on Windows) or the usejdbc2.sh (on UNIX) to switch to JDBC 2.

3. On Windows 2000/NT start the Client Configuration Assistant application.

4. Configure the remote database using the following settings:

    a. Manually configure a connection to a database

    b. TCP/IP Protocol

    c. Host name is the name of the database server

    d. Port number: 50000

    e. Database name: ITSOBANK

    f. Do not set up ODBC datasource (switch off the flag)

5. Before adding the connection, test the connection; use the username/password: dbuser/password.

6. Once the connection is configured and tested, close the Client Configuration Assistant application.

The remote access to the database is configured at this point.

On other systems, where the graphical Client Configuration Assistant application is not available, open a console for DB2 command execution and use the following commands:

```
db2 catalog tcpip node <node_name> remote <remote_hostname> server
<service_name>
db2 catalog db <db_name> as <alias_db_name> at node <node_name>
```

> **Note:** Where
>
> ► <node_name> is the name of the client machine
> ► <remote_hostname> is the name of the remote database
> ► <service_name> is the name associated with the service (find the name in the services file)
> ► <db_name> is the name of the database on the remote machine
> ► <alias_db_name> is the name of the local database, the alias

In case you have any problems setting up the client, refer to the official product documentation of IBM DB2 UDB.

Configuring the user registry for the ITSOBank sample

Setup the following groups and users in your user registry. The user registry can be either local OS, LDAP directory or a custom directory.

Table 12-5   Registry for the ITSOBank sample

Group name	User name
managergrp	manager01
consultantgrp	consultant01
accountantgrp	accountant01
clerkgrp	clerk01
-	mdbuser
-	dbuser
-	jmsuser

You will have to use the user name and the associated password to login to the application.

You have to have the dbuser defined in the local OS user registry; DB2 will do the authentication through the local OS.

You might want to map the groups to the roles defined for the application; instead of the users. It makes the access management easier.

Configuring WebSphere Application Server for the ITSOBank sample

The following procedure will guide you through the installation of the ITSOBank sample.

The sample application requires a couple of settings to be made for the application. To do the configurations follow the steps below:

1. Open the WebSphere Application Server V5 Administrative Console in the browser; *https://yourserver:9090/admin.*

2. Login with the administrator username and password. We are assuming that global security is enabled at this stage. For more information about enabling global security refer to Section 10.2, "WebSphere Global Security" on page 235.

3. Configure the embedded JMS server for WebSphere Application Server.

   a. Select **Servers -> Manage application Servers**.

   b. click the **server1** link on the list of the servers.

   c. Go down on the configuration page and select the **Server Components** link.

   d. Select the **JMSServer** from the components list.

   e. Change the initial state to **Started**.

   f. Add the JMS destination queue for the application to the queueNames list, type in: **itsobankTransferQ** into the text area.

   g. Then click **OK**.

4. **Save** the configuration; note that you will have to confirm the save action.

5. To have the embedded messaging running you have to stop then restart the application server.

6. In order to have the queues working with the embedded JMS server when global security is enabled, authorization has to be set up for the queues.

   a. Open the integral-jms-authorizations.xml file under the <WebSphere install path>\config\cells\<your cell name> then do the following modifications.

   b. Add a user to the <queue-admin-userids> node, for example: `wasadmin`. This has to be a user in the local operating system. At the end, the configuration should look like:

   ```
   <queue-admin-userids>
   ```

```
        <userid>wasadmin</userid>
    </queue-admin-userids>
```

c. Add a queue authorization definition to the root node
   <integral-jms-authorizations>, for the sample application:

```
<queue>
    <name>WS_itsobankTransferQ</name>
    <public></public>
    <authorize>
        <userid>mdbuser</userid>
        <permission>read</permission>
        <permission>write</permission>
    </authorize>
</queue>
```

d. Save the file and close.

7. Create a J2C Authentication entry; select: **Security Center ->
   JAASConfiguration -> J2C Authentication Data Entries**.

8. click **New** to create a new entry, fill out the fields with the following values:

   – Alias: `itsobankds_auth`

   – User ID: `dbuser`

   – Password: `password`

9. Create the following J2C authentication alias

Table 12-6 J2C Authentication Alias

Alias name	username	password
itsobankjms_auth	jmsuser	password

10.Save the configuration changes.

11.Setup a the JDBC provider and the DataSource; select: **Resources ->
   Manage JDBC Providers**

   a. Select the **Server - server1** radio button, then click **Apply**; this will switch
      the scope to the server level.

   b. click **New**.

   c. Select the **DB2 JDBC Provider (XA)** for JDBC Provider. You will need to
      use the XA version of the driver to support the transactions in the
      application.

   d. Click **OK**, then JDBC provider configuration panel appears.

   e. Set the classpath to the correct value to reach the db2java.zip library, for
      example: `c:/sqllib/java/db2java.zip`.

     f. Click **OK**, then the list of the JDBC providers appear on the page.

     g. To create the datasource, select the **DB2 JDBC Provider (XA**) item, you have just created.

     h. Go down on the Configuration page and click the **Data Sources** link.

     i. click **New** to create a new Data source.

     j. Fill out the fields with the following values:

       Name: `itsobankds`

       JNDI Name: `jdbc/itsobankds`

       Use this DataSource in container managed persistence (CMP): Yes. This will create a CMP_Connection_Factory for the WebSphere Relational Resource Adapter with the name: itsobankds_CF, and with the JNDI name: eis/jdbc/itsobankds_CMP.

       Description: `ITSOBank data source`

       Category: `itsobank`

       Container-managed Authentication Alias: `itsobankds_auth` (select from the list).

       Component-managed Authentication Alias: `itsobankds_auth` (select from the list).

     k. click **OK** to add the new data source, then you will see the list of available datasources.

     l. click the **itsobankds** link to open the configuration.

     m. Go down on the page and click the **Custom Properties** link. You will see a list of properties set for the data source.

     n. Change the *databaseName* property with the value: *SAMPLE*; click the link, then change the value to **ITSOBANK**. Click **OK** when done.

     o. Delete the *serverName* and *portNumber* properties.

12. Setup the JMS resources: QueueConnectionFactory and Queue; select **Resources -> Configure WebSphere JMS Providers**

     a. Select the **Server - server1** radio button, then click **Apply**; this will switch the scope to the server level.

     b. click the **WebSphere Queue Connection Factory** link, that will bring up the list of the queue connection factories.

     c. click **New** to add a new factory. The configuration page will appear, then fill out the fields with the following values:

       Name: `itsobankQCF`

       JNDI Name: `jms/itsobankQCF`

Container-managed Authentication alias: `itsobankjms_auth`

Component-managed Authentication alias: `itsobankjms_auth`

Set the connection factory to use transactions: **XA Enabled**.

Leave the rest of the settings as they are.

d. Click **OK** to submit the configuration.

e. Select **Resources -> Configure WebSphere JMS Providers** again, and create a **WebSphere Queue Destination** for the *server1* server. Configure the queue with the following values:

Name: `itsobankTransferQ`

JNDI Name: `jms/itsobankTransferQ`

Leave the rest of the settings as they are.

f. Click **OK**.

13. Setup a ListenerPort for the Message-Driven Bean; select **Servers -> Manage application Servers**.

a. click the **server1** link on the list of the servers.

b. Go down on the configuration page and select the **Message Listener Service** link.

c. On the Message Listener Service page, switch on the Startup box, then click **Apply**.

d. Select the **Listener Ports** link, which will show the list of the available listener ports. Add a new port by clicking on the **New** button, then fill out the fields with the following values:

Name: `IncomingTransferPort`

Initial state: `Started`

Connection Factory JNDI Name: `jms/itsobankQCF`

Destination JNDI Name: `jms/itsobankTransferQ`

e. Click **OK** to submit the configuration.

14. Save the configuration before installing the ITSOBank sample application. click **Save** on the Administrative Console menubar, then confirm it again with **Save**.

15. Install the ITSOBank sample application.

a. Select the **Applications -> Install New Application** item.

b. Browse for the **itsobank.ear** archive on the system, then click Next. Wait until the file is loaded, an animated progressbar appears during the process.

Appendix A. Sample application   **453**

    c. Navigate through the installation pages clicking **Next** on the pages.

    d. When you get to the Role Mapping page, map the users or groups to the roles. The following roles have to be mapped:

- **manager** role maps to **managergrp** group

- **accountant** role maps to **accountantgrp** group

- **consultant** role maps to **consultantgrp** group

- **clerk** role maps to **clerkgrp** group

- **mdbuser** role maps to **mdbuser** user

    Note that the allauthenticated role is already mapped to the AllAuthenticated special role.

    e. click **Finish** on the last page, that will start installing the ITSOBank sample application. The animated progressbar appears during the process again.

    f. Wait until the application gets installed and the report page appears. Go down to the end of the report page and select the **Save Configuration** link. Confirm the save again with the **Save** button on the next page.

    g. Now the application is saved, but not ready to start.

16. Select **Environment** from the navigation menu, then regenerate the plugin for the HTTP server.

17. Logout.

To make the changes effective stop the server by issuing the command: `stopserver server1` under the WebSphere Application Server V5 base *bin* directory, then starting it with the `startserver server1` command.

> **Note:** for errors look at the SystemOut.log file under the WebSphere Application Server V5 base *logs/server1* directory.

Importing the sample application into the development environment

The ITSOBank sample application provided together with the book contains not only the compiled code for the runtime but the source code for the development environment as well.

1. After you decompressed the additional material you will find the itsobank.ear enterprise archive. Start your WebSphere Studio Application Developer V5 with a new project folder, where you can import the ITSOBank sample application.

2. Open or switch to the J2EE perspective then select **File -> Import** from the menu.

3. Select the **EAR file** from the Wizard, then click **Next**.

4. On the next panel browse for the itsobank.ear file, find the directory where you have unpacked the additional material. Provide a project name, for example: `itsobank`. Click **Next**.

5. Click **Next** again, unless you want to change the project location for the import.

6. The next panel is only information about the EAR file, click **Next**.

7. On the last panel click on **Finish** and waint until the project gets imported, it might take several minutes.

Optionally you can import a couple other projects onto your workspace, they are:

► itsobankCustomRegitry.zip is an implementation of a custom user registry using the DB2 database.

► itsobankThinClient.zip is a thin Java client to access application functions from a client machine.

► TrustAssociationInterceptor.zip is an implementation of a trust association interceptor.

► IDAssertion.ear is a test application for identity assertion, and requires three machines for the runtime.

12.9 Where to find more information

For more information about the sample application, refer to the different chapters and sections in the book, that discusses a particular function or module.

You can also find detailed information in the source code of the application.

B

LDAP configurations

This appendix covers step-by-step configuration settings for different LDAP servers to use with WebSphere Application Server V5.

Each LDAP configuration consist two scenarios, one with non-secure directory access and one with secure directory access, using SSL. Microsoft Active Directory is an exception there is no SSL configuration for that scenario.

The directory servers described here are:

► Lotus Domino Server

► Netscape's iPlanet Directory Server

► Microsoft Active Directory Server

SecureWay Directory Server

The configuration WebSphere using the SecureWay Directory Server V3.2.2 as LDAP user registry is covered in Section 10.13, "Connecting to directory servers (LDAP)" on page 315.

IBM Directory Server

IBM Directory Server 4.1 does not differ much from the SecureWay Directory Server V3.2.2. The configurations for IBM Directory Server are the same as for the SecureWay Directory Server in Section 10.13, "Connecting to directory servers (LDAP)" on page 315.

Lotus Domino

This section covers the LDAP configurations for Lotus Domino V5.06a.

Configuring Websphere to use Domino LDAP

In order to set up WebSphere security settings using the Domino Directory as the LDAP server, it is necessary to specify a Security Server ID, that is, the user ID under which the server runs for security purposes. This user ID should be any user registered in the Domino Directory (Notes or Internet/intranet users) with an Internet password set. We recommend that you create a new specific user in the registry to be used by WebSphere.

For that purpose, start the Domino R5 Administration with a notes ID having at least author access for creating new documents in the Domino Directory; make sure the *UserCreator* role is selected and follow these instructions:

1. Open the Domino Server from the left server bookmark pane and select the **People and Groups** tab.

2. Click the **Add Person** button.

3. Add a new user as shown in figure below with the Short Name/UserID and Internet password set.

Figure 12-30   Websphere Administration Person document in Domino Directory

4. Save and Close the document.

Before configuring WebSphere to use domino LDAP make sure that LDAP task is running on Domino Server. In order to check it, from the Domino Administrator select **Server** tab and then **Status**. You should see the window similar to the one below.

Figure 12-31   Checking for Domino LDAP server task

You can alternatively issue `show task` command at the server console and look for LDAP task. If the task is not running issue the command `load LDAP` at the console or modify **ServerTasks** line in server notes.ini file to include LDAP task, and restart server.

Ensure that Domino using the LDAP protocol and users are listed in the direcotry and can be found under the right suffixes. Domino provides a command-line search utility that allows you to use LDAP to search entries in the Domino Directory on a server that runs the LDAP service, or search entries in a third-party LDAP directory.

This tool is included in the Domino server and Notes client software.

> **Note:** To use the *ldapsearch* tool for searching against a Domino Directory, the LDAP task in the Domino Server must be started and the notes.ini file must be included in the machine system's Path environment variable where ldapsearch will be executed.
>
> To search for wasadmin user in Domino LDAP issue the following command at the command prompt:
>
> ```
> ldapsearch -v -h <your ldap server hostname> "uid=wasadmin"
> ```

Configuring WebSphere to use Domino LDAP

To configure WebSphere to use Domino as its user registry, follow the steps below.

1. Start the WebSphere Administrator's Console.

2. Expand the tree **Security -> User Registries -> LDAP**. You will see the LDAP configuration panel open in the main window.

3. Fill in the following configuration settings:

   Server User ID: this field must contain the value specified in the Short Name/User ID field in the Person Document of the Domino Directory created in the steps above for the WebSphere administrator. This is the user ID that will have to be used for login to start the WebSphere Administrator's Console once security is enabled, for example: `wasadmin`.

   Server User Password: enter the Internet password set for the wasadmin user in this document.

   Type: `Domino`

   Host: name for the Domino (directory) server, for example: `dominosrv`

   Port: `389`

   Base Distinguish Name: this is the base distinguished name of the directory service, indicating the starting point for LDAP searches of the directory service. As we defined all our users and groups under /ITSO, we have entered `o=itso` for this field.

> **Note:** Leaving this field empty, in our scenario resulted a Java exception while performing searches for users and groups to mapped to application roles. The exception message that appeared was:
>
> ```
> [8/15/02 19:25:03:303 EDT] 2c771047 SecurityAdmin E SECJ0234E: An
> unexpected exception occurred when trying to get groups from the User
> Registry with pattern * and limit 20. The exception is
> com.ibm.websphere.security.CustomRegistryException: null: [LDAP: error
> code 34 - The search base null appears to have an incorrect syntax]
> ```

Bind Distinguish Name: distinguished name used when WebSphere binds with the Domino server. If no name is specified, the administration server will bind anonymously.

Bind Password: if a user is specified in the Bind Distinguished name, include the corresponding password here.

Other fields you may leave default. Our settings from the ITSOBank scenario are presented in the picture below.

Figure 12-32   Domino LDAP Settings in WebSphere Administrative Console

4. Click **OK** at the bottom of the configuration panel.

5. Save the configuration for WebSphere.

6. Now you need to enable LDAP user registry in Global Security section; select **Security -> Global Security**.

7. **Enable** global security, select **LDAP** for the Active User Registry**.**

8. Click **OK** at the bottom of the configuration panel, WebSphere will validate the server user against the user registry.

9. Save the configuration WebSphere.

10. You will now need to restart your WebSphere Application Server to make the changes effective. Logout from the Administrative Console and restart your WebSphere Application Server.

11. When you log on to the Administrative Console for the next time you will be prompted for user name and password. Enter the values that you have provided in LDAP configuration panel in **Server User ID** and **Server User Password**.

Your WebSphere is now configured to use Domino LDAP directory.

Configuring Websphere for secure LDAP connection

This section describes the steps to perform in order to secure the connection between the WebSphere Application Server and the Domnio LDAP server.

Configuring Domino to use SSL

To configure Domino for secure communication for LDAP, Domino requires a keyring, .kyr file, and password stash, .sth file, on the server side. In order to create the keyring use Domino's built in Certificate Authority (CA) application.

1. The CA application installs with Domino, you should be able to find it under the directory c:\lotus\domino\data. Open the database in Lotus Notes.

> **Note:** Do not open the CA application in the administation application, because it will not work.
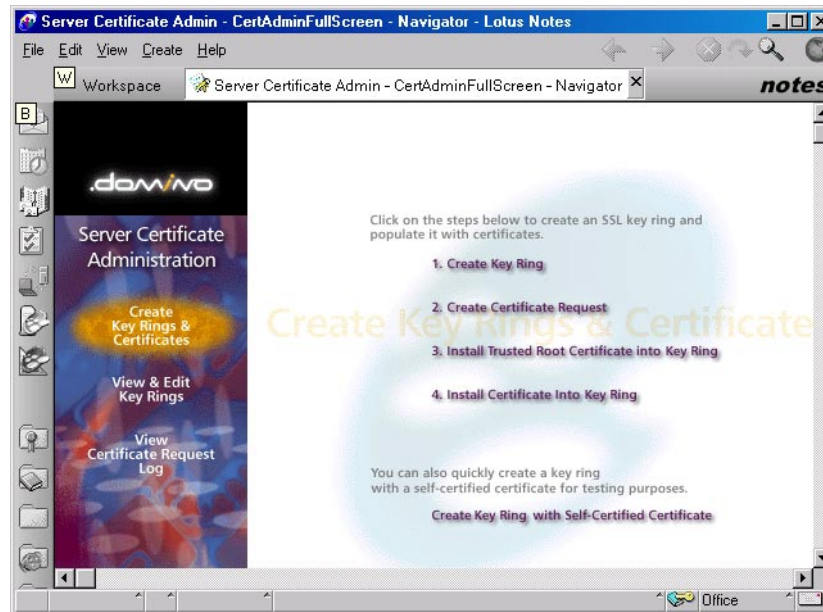
Figure 12-33   Domino CA application

2. Select the **Create Key Ring with Self-Certified Certificate**.

3. Fill out the certificate request as indicated on the following screen.

> **Important:** Make sure that the keyfile and the certificate is saved under the Domino data directory, for example: c:\lotus\domino\data.
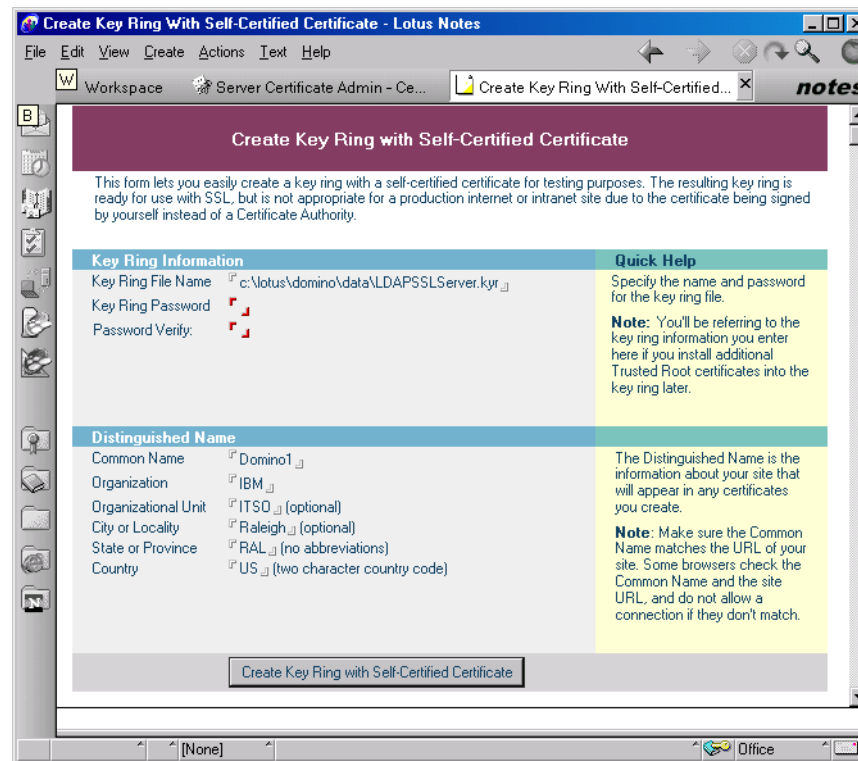
Figure 12-34   Creating a self-certified certificate

4. click the **Create Key Ring with Self-Certified Certificate** and Domino will generate the keyring with the stash file and save it in the Domino data directory.

5. Close the CA application in Lotus Notes.

6. Start the ikeyman tool that comes with the IBM HTTP Server, capale of reading the .kyr files, and open the keyring file you have just created: LDAPSSLServer.kyr.

7. Export the Key Pair certificate under the name: `LDAPSSLServer.arm`, to the Domino data directory, `c:\lotus\domino\data`.

8. Close the ikeyman application.

Enable SSL on Domino Server

Enable SSL for the Domino server by modifying the **Server** document stored in the server's **Notes Name & Address Book**.

1. Make sure you have the keyring file, `LDAPSSLServer.kyr`, and password stash file, `LDAPSSLServer.sth`,in the server's data directory, `c:\lotus\domino\data`.

2. Launch **Domino Administrator**, and make sure that **Use directory on** field point to your Domino Server.

3. Select **Configuration** tab.

4. On the left panel expand **Server** section and click **Current Server Document**. You will be presented with server document from your server's Name and Address Book.

5. Click **Edit Server** and select the **Ports** / **Internet Ports** tab.

6. In the **SSL key file name** field put your key ring file name that you have created before. In our case the file name was `LDAPSSLServer.kyr`

7. click the Directory tab and modify the following fields:

   SSL port number: `636`

   SSL port status: `Enabled`

8. Other fields you can leave default. Picture below presents our server document from the scenario tested in this book.

Figure 12-35   Domino LDAP SSL configuration

9. Click **Save and Close** on the top of the server document.

10. You should restart your server in order for the changes to take effect.

After your server restarts you can verify if the LDAP server is listening on port 636. After issuing the command `sh ta` at the server console a message appears, similar to the following one.

Look for the line on the console:

```
LDAP Server     Listen for connect requests on TCP Port:389 SSL Port:636
```

This line will tell you that the LDAP server is running and listens on the non-secure and on teh secure port.

Configuring WebSphere to use SSL

WebSphere Application Server has to have the LDAP server certificate, in ordert to participate in the SSL connection. For this reason, you have to import the LDAP server's SSL certificate into WebSphere's server trust file.

1. Open the ikeyman tool that comes with WebSphere, capable to handle the .jks files, open the server trust store file, if you are using the dummy keystore, open the <WebSphere_root>\etc\DummyServerTrustFile,jks.

2. Import the LDAPSSLServer.arm as a signer certificate; use the file from the Domino server, you will have to copy the .arm file to your WebSphere server machine.

3. Close the ikeyman utility.

To create a new SSL entry and configure WebSphere to use it to connect to the LDAP server, follow the steps from Section , "Configuring secure LDAP (LDAPS) connection" on page 325 using the following information.

iPlanet Directory Server

In this section, we will cover the steps required to configure WebSphere with NetScape iPlanet Directory Server V5.0. In this scenario, we have installed Access Manager using the native installation method.

Configuring WebSphere to use iPlanet Directory Server

In order to configure WebSphere's access to iPlanet Directory Server, we must first define a user entry for WebSphere to use for binding to the directory, as we did for IBM Directory Server.

The only change we have made is that we are now using a directory suffix of `o=tamral,c=us` instead of o=itso.

After you have created your user entry, WebSphere is ready to be configured to use iPlanet Directory Server as it's user registry.

1. Start the WebSphere Administrator's Console. Once you have started the console, login, and select **Security -> User Registries -> LDAP**. This will display the LDAP User Registry panel.

2. Fill out the LDAP configuration page as follows:

   Server User ID: enter either the fully qualified Distinguished Name (DN) for the WebSphere server ID user, we used the DN: `cn=wasadmin,o=tamral,c=us`.

   Server User Password: enter the password for your user ID.

   Type`: Netscape`.

   Host`:` enter the fully qualified DNS name of your iPlanet Directory Server. In our configuration, our host name is `tivoli9.svo.dfw.ibm.com`.

   Port: `389`

   Base Distinguished Name: enter the suffix under which your user entries for WebSphere will be defined in your directory. In our configuration, this was `o=tamral,c=us`. This value is the base from which all user searches will be conducted.

   Click **OK** at the end.

LDAP User Registry

LDAP User Registry settings are used when users and groups reside in an external LDAP directory. When security is enabled and any of these properties are changed, please go to the GlobalSecurity panel and click Apply or OK to validate the changes. ⓘ

Configuration

General Properties

Server User ID	★ cn=wasadmin,o=tamral,c=us	ⓘ The user ID under which the server will execute (for security purposes).
Server User Password	★ ********	ⓘ The password corresponding to the serverId.
Type	iPlanet ▼	ⓘ The type of LDAP server being connected to.
Host	★ tivoli9.svo.dfw.ibm.com	ⓘ Specifies LDAP server host name.
Port	389	ⓘ Specifies LDAP server port.
Base Distinguished Name (DN)	o=tamral,c=us	ⓘ The base distinguished name of the directory service, indicating the starting point for LDAP searches of the directory service.
Bind Distinguished Name (DN)	cn=root	ⓘ The distinguished name for application server to use to bind to the directory service.
Bind Password	****************	ⓘ The password for the application server to use to bind to the directory service.
Search Timeout	120	ⓘ Specifies the timeout value in seconds for an LDAP server to respond before aborting a request.
Reuse Connection	☑	ⓘ Should set to checked by default to reuse the LDAP connection. Set to unchecked only in rare situations where a router is used to spray requests to multiple LDAP servers and when the router does not support affinity.
Ignore Case	☑	ⓘ When set to true, a case insensitive authorization check will be performed.
SSL Enabled	☐	ⓘ Whether secure socket communications is enabled to the LDAP server. When enabled, the LDAP Secure Socket Layer settings are used if specified.
SSL Configuration	DefaultSSLSettings ▼	ⓘ Specifies the LDAP SSL Settings configuration setting.

Apply   OK   Reset   Cancel

Additional Properties

Advanced LDAP Settings	Advanced LDAP User Registry settings are used when users and groups reside in an external LDAP directory. When security is enabled and any of these properties are changed, please go to the GlobalSecurity panel and click Apply or OK to validate the changes.
Custom Properties	A set of arbitrary user registry configuration properties whose names are specific to a given type of pluggable registry.

Figure 12-36   LDAP configuration for iPlanet LDAP server

3. WebSphere will validate your entries and display the Global Securty page, **Enable** global security.

4. Scroll down the Configuration panel and select **LDAP** for the Active User Registry selection.

5. Click the **OK** button. Once WebSphere validates your configuration, and returns with no errors, save your current configuration. You will then need to stop and restart your WebSphere server before proceeding.

Once you have restarted your WebSphere server, open the WebSphere Administrative Console in your browser. You will now see, in addition to a user ID field, a Password field is displayed. To login, use a valid user ID and password. You have now successfully configured WebSphere to use the NetScape iPlanet Directory Server as its user registry.

> **Note:** In our case, we used the user ID and password we defined for the Server User ID on the LDAP User Registry's panel . Under the Security Center, you could select Manage Console Users to define additional user ID's, however, they must first be defined in your directory server.

Configuring WebSphere SSL access to iPlanet Director Server

Now that we have configured WebSphere, we will now proceed, as we did with the IBM Directory Server, to secure our connection using SSL. As with the IBM Directory Server scenario, we need to establish a trusted relationship between WebSphere and our iPlanet Directory Server. Unlike IBM Directory Server, however, the iPlanet Directory Server does not allow us to generate a self-signed key for use with its server. We will have to obtain a server certificate from a Certificate Authority (CA). In our scenario, we used our own CA to generate the certificates being used. In your environment, you may well be using a commercial CA to accomplish this task. Regardless, the steps are basically the same; only the details as to how to obtain your certificate, and your root CA certificate will differ.

Obtaining a server certificate for iPlanet Directory Server

In order to obtain a digital certificate for our iPlanet Server, we must first build a certificate request to send to a CA. This certificate request will contain the identity information of our iPlanet Directory Server, as well as the public key for the corresponding private key that the Directory Server will generate for our request. The process of obtaining a certificate from our CA is what establishes trust; the CA will verify by some means the identity of the requestor for a certificate, and will sign the certificate, establishing its authenticity. Thereafter, any entity which receives the certificate, can, by virtue of the signing CA, establish that the certificate is indeed valid, presuming of course that the recipient also trusts the signing CA.

To request, and generate, a server certificate for iPlanet Directory Server, follow these steps:

1. Open your iPlanet console, and highlight your iPlanet server as shown below:
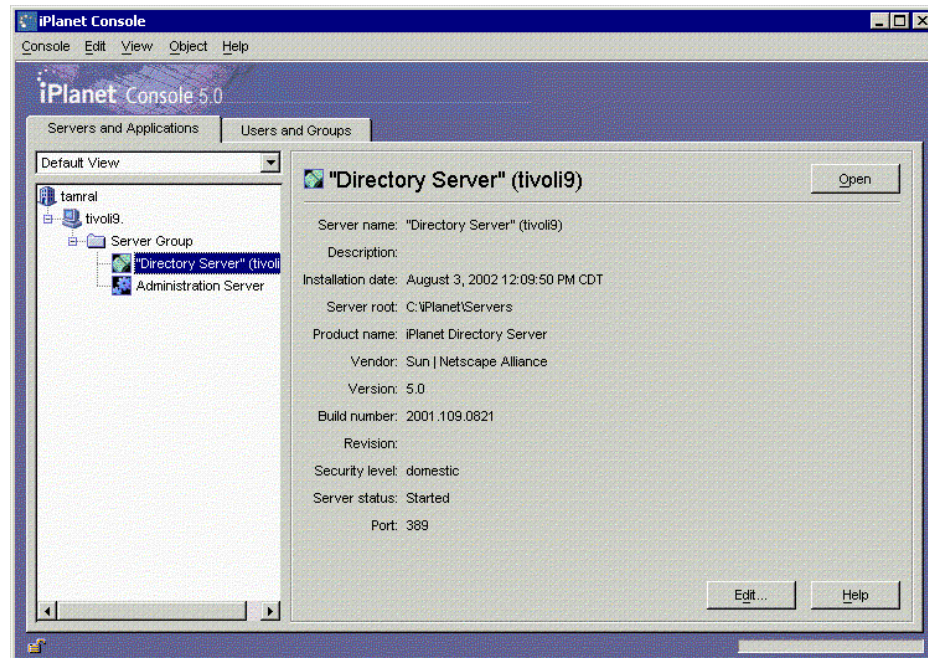
Figure 12-37   iPlanet Directory Server console

2. On the upper right side of the pane, click the **Open** button.

3. The Directory Server Management panel will display. On this panel, click **Manage Certificates** to display the Manage Certificates panel. If this is the first time you have selected this task, the Set Security Device panel will display. At this time, chose a password to protect your directory server keystore. And be sure you remember it.
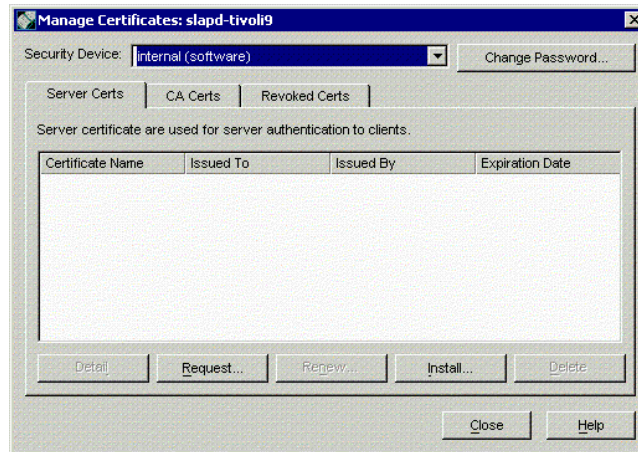
Figure 12-38   iPlanet Directory Server Manage Certificates panel

4. Select the **Server Certs** tab as shown, and then click the **Request...** button. This will start the Certificate Request Wizard.

5. We are going to request our certificate manually. click the **Request certificate manually** button as shown below:
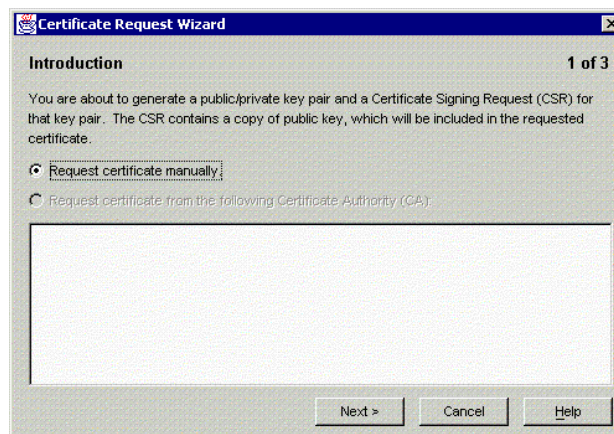
Figure 12-39   iPlanet Certificate Request Wizard panel 1

6. click the **Next >** button. The second panel of the wizard, Requestor Information, will then display, as follows.
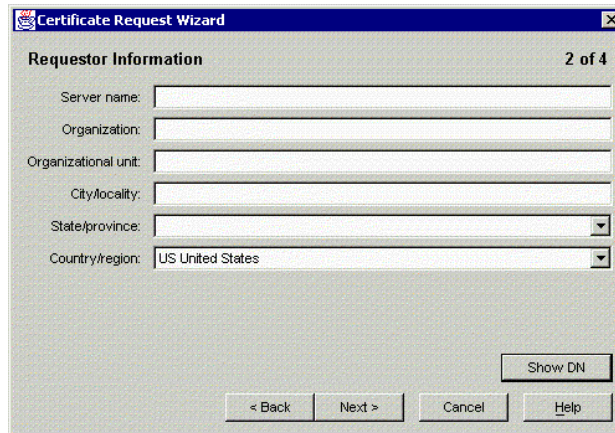
Figure 12-40   iPlanet Certificate Request Wizard panel 2

7. On this panel, we will enter the information required by our CA to identify our directory server. The actual values you enter will be specific to your CA. The Server name field is required, and must match the DNS name of your director server. This name must be the fully qualified DNS name, and not just the host name. Depending on your environment, enter the values required and when finished, click **Next >**.
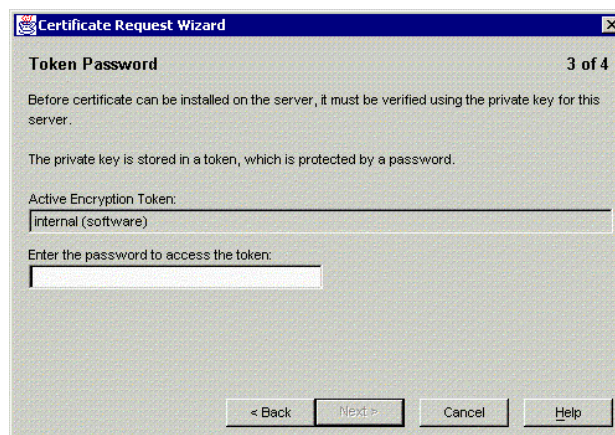
Figure 12-41   iPlanet Certificate Request Wizard panel 3

8. The Token Password panel will then be displayed. in the Enter the password to access token field, enter a password to use to install your certificate once you have received it from your CA. After entering a password, click the **Next >** button, and the following panel will be displayed.
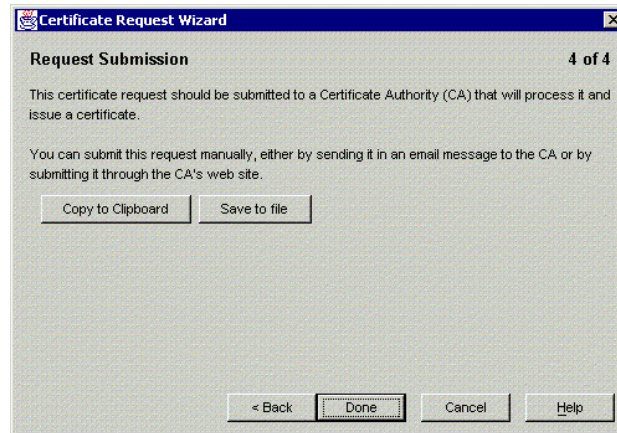
Figure 12-42   iPlanet Certificate Request Wizard panel 4

9. On the Request Submission panel, you can select to either copy the certificate request to the clipboard, or save the request to a file. Depending on your CA, you will chose one or the other option. The CA we are using requires us to paste the certificate request into a browser window, so we chose **Copy to Clipboard**. After storing your certificate request in the format you require for your CA, click **Done**. This will close the Certificate Request Wizard.

10. After saving your certificate request, you now need to send the request to your CA. As we stated earlier, the CA we are using uses a browser interface to paste the request into. Some CA's will require you to email the request. In either case, at some point the CA will generate a public certificate for our request, and will have some process for obtaining the certificate from them. In our case, we downloaded the certificate from our CA once it was approved. Some CA's will send an email. In either case, once we receive our certificate back from the CA, we next need to install it. To do this, we must return to the Manage Certificates panel for our iPlanet Directory Server. Once there, click the **Install..** button. The first panel of the Certificate Install Wizard will then display, as shown below.
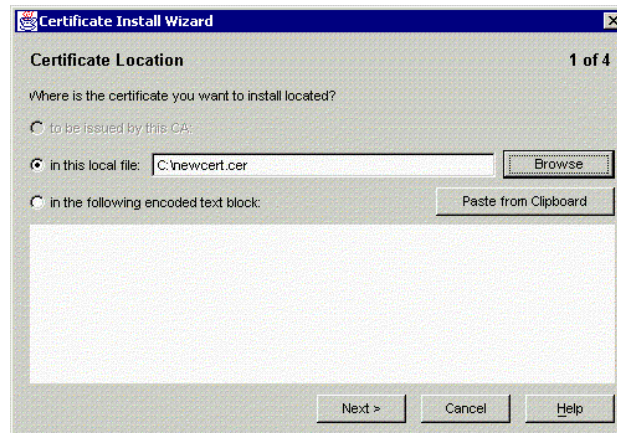
Figure 12-43   Certificate Install Wizard panel 1

11. To install our certificate, we must first specify the location. In our environment, we download the certificate, so we chose in this local file. If you have placed your certificate into the clipboard, then select in the following encoded text block, and click the **Paste from Clipboard** button. Once you have entered the certificate location or pasted it into the panel, click the **Next>** push button, and the following panel will be displayed.
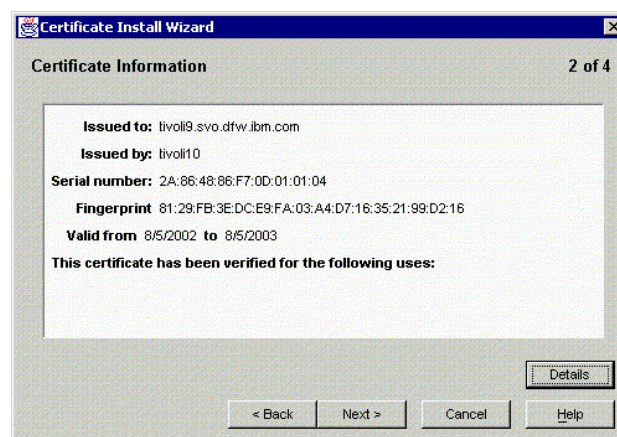
Figure 12-44   Certificate Install Wizard panel 2

12. On this panel, the details of our new certificate are displayed. Note that the certificate is issued to our iPlanet Directory Server, and that it has been issued by our CA. click the **Next>** button, and the third panel of the wizard will display as follows.
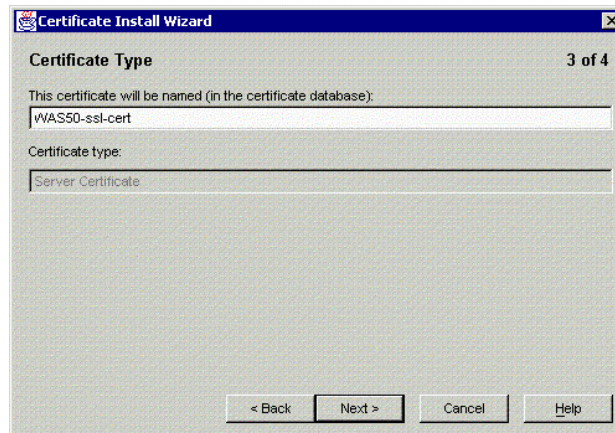
Figure 12-45   Certificate Install Wizard panel 3

13. On this panel, we need to enter a name for the certificate. The name entered will be used in the iPlanet Directory Server certificate database to identify this certificate. We have chosen to name our certificate **WAS50-SSL-cert**. Note that the Certificate type is Server Certificate. Once you have entered the a name for your certificate, click the **Next>** button, and the next panel will display.
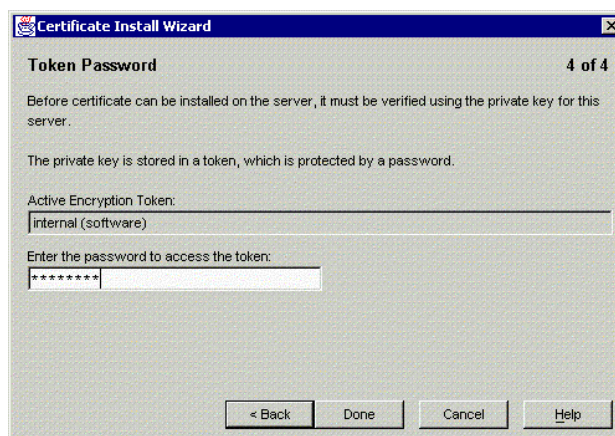
Figure 12-46   Certificate Install Wizard panel 4

14. On this panel, we need to enter the token password we specified above when we made our certificate request. In the Enter the password to access the token: field, enter the password you specified, and then click the **Done** push button to complete the certification installation. The Manage Certificates

panel, shown below will then be redisplayed. The new certificate we just installed is now shown, displaying the certificate name we specified above.
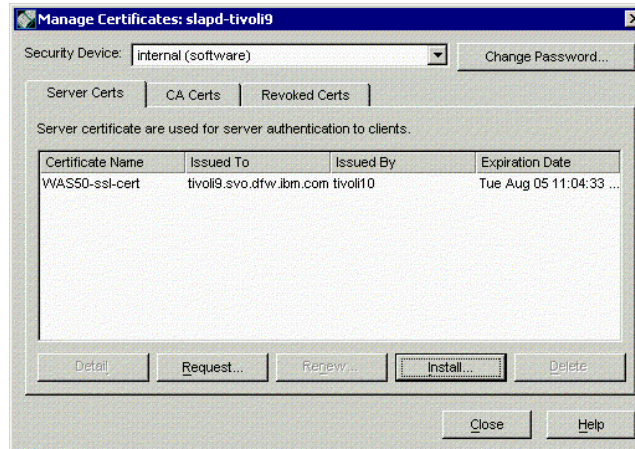
Figure 12-47   Manage Certificates panel with installed certificate

Now that we have our server certificate for our directory server, we are ready to begin configuring WebSphere to use an SSL connection to our iPlanet Directory Server.

Configuring iPlanet Directory Server for SSL access

Wow, was that a lot of work. But it was worth it. We are now ready to configure our iPlanet Directory Server for SSL access. To begin, select the Configuration tab on your iPlanet console. Then on the configuration panel, select the Encryption tab. The following panel will be displayed.
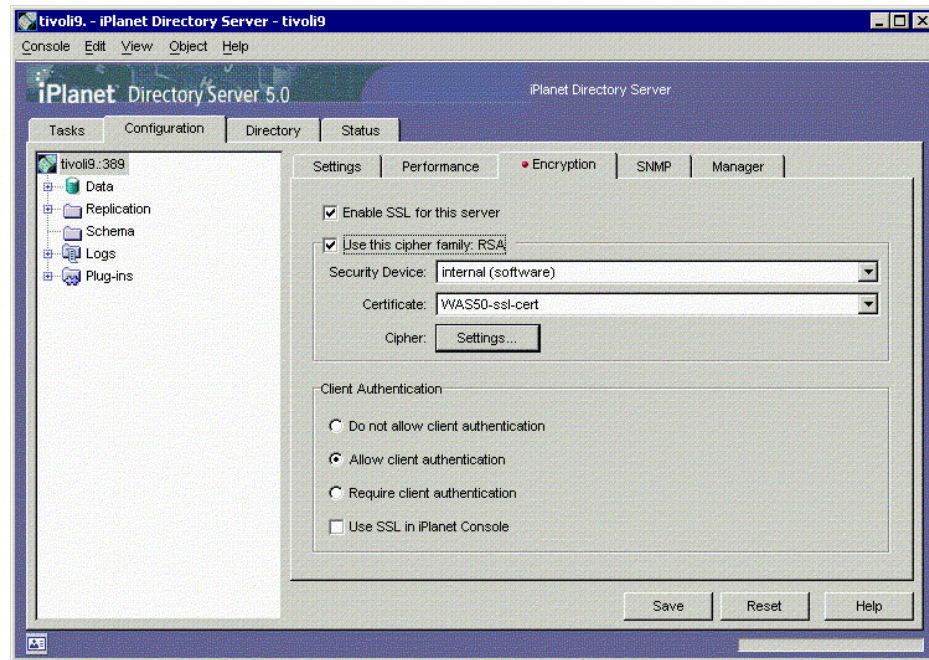
Figure 12-48   iPlanet Encryption settings

On this panel, we will configure our iPlanet Directory Server to allow SSL access. Select the **Enable SSL for this server** check box. Also select the **Use this cipher family RSA** check box. In the Security Device: field, ensure that the security device in which you installed your server certificate is selected. We used the default device, **internal (software)** as shown. For the Certificate field, we need to select the server certificate we just finished installing. In our case, that was **WAS50-SSL-cert**. Now, click the **Settings** button, and the Cipher Preference panel will be displayed as shown below.
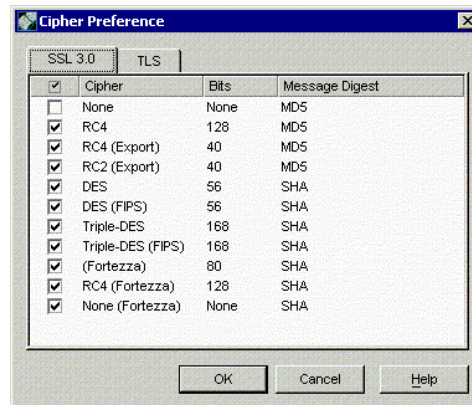
Figure 12-49   Cipher Preference panel

Select the **SSL 3.0** tab, and verify the cipher keys are selected as shown above. Then click the **OK** button. The Encryption settings panel will then be redisplayed. The Allow client authentication radio button under Client Authentication is selected by default, which we accepted for our configuration. Once you have completed your configuration settings, click the **Save** button. We have now configured iPlanet Directory Server to allow SSL access.

WebSphere SSL configuration

We are now ready to begin the configuration of WebSphere for SSL access to our iPlanet Directory Server. To begin, we must first set up the trust relationship to our directory server. To do this, we are going to set up a new keystore file for use by WebSphere. In this keystore, we are going to add the CA certificate from the CA that issued the server certificate for our iPlanet Directory Server as a trusted signer. When we do this, it means that for any certificate that WebSphere receives using this keystore, it will trust that certificate because it is signed by our CA.

Before proceeding, we need to obtain the public certificate from our CA. Depending on the CA you are using, the method used will vary. In our case, we had the option to download the CA using a browser, and saved it on our WebSphere system. You will need to do the same before proceeding. Note that some CA's will give you an option as to what format to use when obtaining its certificate. In this case, request the **Base64-encoded** format.

Appendix B. LDAP configurations   **479**

> **Tip:** from a battle scarred veteran protigy "grasshopper".
>
> You really, really need to create a new keystore file. If you don't, you will get to spend this Friday night in the lab, like those poor IBM Directory Server folks who ignored my tip got to do last Friday night. Of course, you didn't read my tip to them, did you? That makes perfect sense, as you are working with iPlanet Directory Server. May I suggest that you refer to my tip in their section. Then you can decide how you want to spend this Friday night.

To create our keystore file, follow the steps from Section , "Generating a self-signed certificate for the Web Container" on page 302.

Create a new key database using the following information:

> Key database type: `JKS`
>
> File name: `iPlanet.jks`
>
> Location: `/usr/WebSphere/AppServer/etc`

Import the certificate using the following information:

> Certificate format: `Base64-encoded ASCII data`
>
> Certificate file name: `export_Tivoli10.cer`
>
> Location: `/tmp`

Now that we have our keystore file, we are now ready to begin the configuration of WebSphere for SSL access to our directory server. To begin, login to your WebSphere Administrative Console from your browser, and follow these steps.

1. The first thing we need to do is to define the SSL settings to use our keystore file. Follow the steps from Section 10.8.1, "SSL configurations" on page 258 using the information below:

   Alias: `iPlanetssl`

   Key File Name: `/usr/WebSphere/AppServer/etc/iPlanetkey.jks`

   Key File Password: type in the password for the key file

   Key File Format: `JKS`

   Trust File Name: `/usr/WebSphere/AppServer/etc/iPlanetkey.jks`

   Trust File Password: type in the password for the key file

   Trust File Format: `JKS`

   Security Level: `High`

2. Select **Security -> User Registries -> LDAP** and provide the following information for the LDAP configuration:

   Server User ID: enter the DN for your WebSphere server id. In our example, this is `cn=wasadmin,o=tamral,c=us`.

   Server User Password field: enter the password for your server id.

   Type: `Netscape`

   Host: enter the DNS name of your iPlanet Directory Server. In our example, we entered `tivoli9.svo.dfw.ibm.com`.

   Port: `636`

   Base Distinguished Name (DN): enter the directory suffix under which your WebSphere users are stored in your directory server. In our example `o=tamral,c=us` as our suffix.

   Select the `SSL Enabled` checkbox.

   SSL Configuration: `iPlanetssl`

   After finishing your entries, click **OK**.

3. The global security settings will appear, check the **Enable** button for security. Select **LDAP** in the Active User Registry field, in case you have not done yet. Click **OK**.

4. Once WebSphere validates your new configuration, you will need to save the configuration, and restart your WebSphere server. If any errors are found, go back and check your entries before saving.

Once you have restarted your WebSphere server, you will now be using SSL to communicate with your iPlanet Directory Server.

Microsoft Active Directory

In this section, we will cover the steps required to configure WebSphere Application Server V5 to use Microsoft Active Directory as its user registry. To use Active Directory, you must have a Windows 2000 domain in your environment. To configure your WebSphere server to use Active Directory as its user registry, follow these steps.

1. To begin, start the Active Directory Administration console by selecting **Start -> Programs -> Administrative Tools -> Active Directory Users and Computers**. The following panel will display.
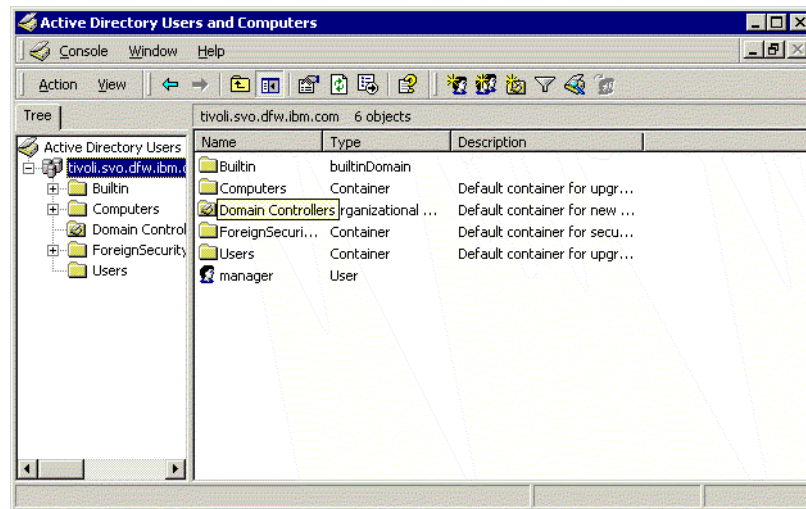
Figure 12-50   Active Directory Users and Computers console

2. On the left pane, select the Active Directory domain for you environment. In our scenario, this is **tivoli.svo.dfw.ibm.com**. We are now going to create an organizational unit. Select **Action -> New -> Organizational Unit**. The following panel will be displayed.
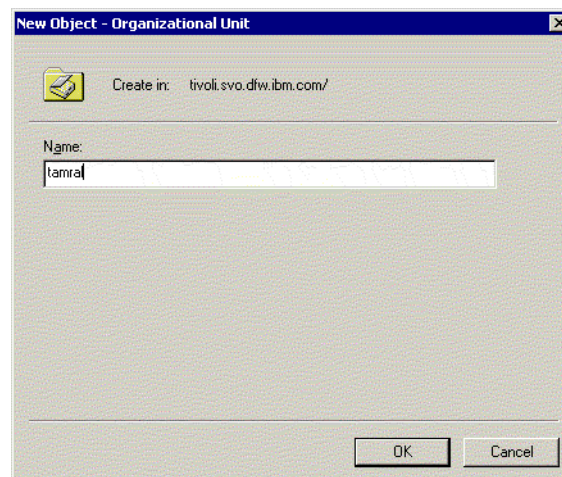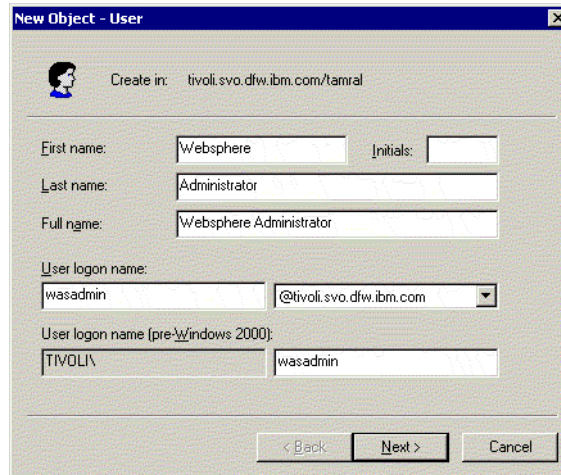
Figure 12-51   Active Directory New Ob ject panel

3. Enter the name for your organizational unit. In our example, we used **tamral**. click the **OK** button. The Active Directory console will then be displayed, and our new organizational unit will now be listed in the left pane. To add the

WebSphere administrators id to Active Directory, select our new organizational unit. Then select **Action -> New -> User**. The following panel will then be displayed.

Figure 12-52   Active Directory New User panel

4. On this panel, we will define our administrator ID for WebSphere. In the First name and Last name fields, enter a value to identify this id as the WebSphere administrator id. We used `Websphere` for First name and `Administrator` for Last name. In the User logon name field, enter the id for the WebSphere administrators ID, we used `wasadmin`. After making your entires, click the **Next>** button. The following panel will be displayed.

Figure 12-53   New User password panel

5. On this panel, enter the password for the wasadmin ID in the Password and Confirm password fields, and select the **Password never expires** check box. After completing your entries, select the **Next>** button.

6. On the next panel, click the **Finish** button. Our WebSphere administrator ID has now been created.

To view our new user, refresh the Active Directory console panel, and highlight our organizational unit, **tamral**. The following panel will be displayed.
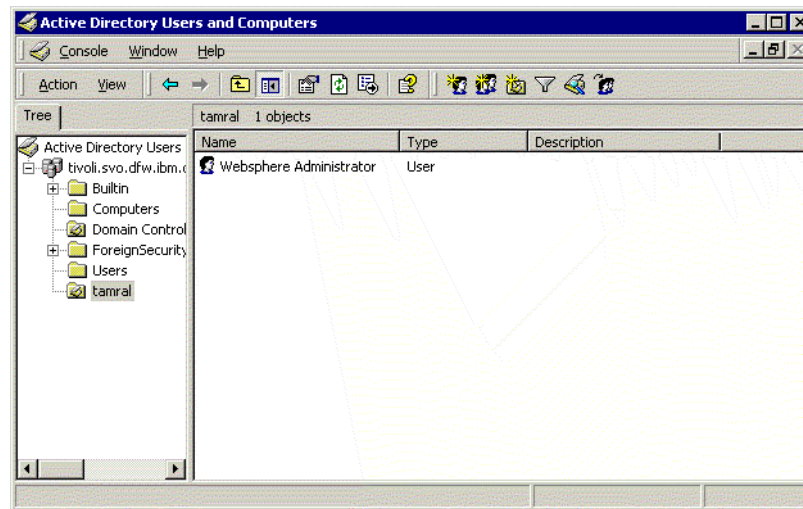
Figure 12-54   New Websphere administrator user in Active Directory

Configuring WebSphere to use Active Directory

Now that we have created our WebSphere administrator ID, we can configure WebSphere to use Active Directory as its user registry. To configure Active Directory as the user registry for WebSphere, follow these steps.

1. Start the WebSphere Administrators console, and login. Select **Security -> User Registries -> LDAP**.

2. Provide the following information on the LDAP configuration page:

   Server User ID: enter your Active Directory user id. We entered `wasadmin` (this is the same ID created in the Active Directory registry in the previous section).

   Server User Password: enter the password for your administrator ID.

   Tyep: `Active Directory`

   Port: `389`

   Base Distinguished Name: enter the location within Active Directory that your WebSphere users will be defined. For Active Directory, you must use this field to define the starting point for WebSphere to use when doing searches. In our example, we entered `dc=tivoli,dc=svo,dc=dfw,dc=ibm,dc=com`, the base suffix for our Windows domain `tivoli`.

Bind Distinguished Name: enter the DN of your WebSphere administrator user. This field is required when using Active Directory as a user registry. By default, Active Directory does not allow anonymous users to access group membership, and other group and user information which WebSphere requires access to. For our administrator, the fully qualified DN we entered is `cn=Websphere Administrator,ou=tamral,dc=tivoli,dc=svo,dc=dfw,dc=ibm,dc=com.`

Bind Password: enter the password for your WebSphere administrator.

Click **OK**.

3. The Global Security panel will be displayed, **Enable** global security, scroll down and in the Active User Registry field, select **LDAP**.

4. Click the **OK** button.

5. Your updates will now be validated, if no errors are found, save your configuration.

6. Restart your WebSphere server, and start the WebSphere Administrative Console. You will now be prompted to enter a user ID and password.

You have now completed the configuration of WebSphere using Active Directory as its user registry.

Testing LDAP connections

There are cases when you can run into difficulties when configuring WebSphere to use LDAP directory as a user registry. In these cases the first step is to isolate the problem by testing your LDAP connection.

For testing you should try to connect to the LDAP server from the WebSphere machine. Test the LDAP connection without security, then test the connection with security (LDAPS) if you are planning to use that.

For testing purposes you can use any LDAP client on the server machine, `ldapsearch` command line utility from the IBM Directory Server distribution (part of the client), ldapsearch command line utility from Lotus domino, Netscape or Mozilla's address book search, or Microsoft Outlook (Express) address book search.

When you are testing the connection make sure you provide the binding dn with the password. When testing the secure connection (LDAPS) make sure you have the LDAP server's public certificate installed or provided as a paramater for the client; if client-side authentication is configured, import the client certificate into the LDAP server's keyring.

C
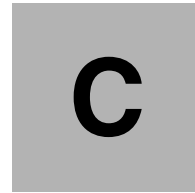
Single Sign-On with Lotus Domino

This chapter discusses Single Sign On Scenarios between IBM WebSphere Application Server 5.0 and Lotus Domino Application Server. Scenarios presented here include:

► WebSphere Application Server and Lotus Domino Application Server based on user registry stored in IBM SecureWay Directory

► WebSphere Application Server and Lotus Domino Application Server based on user registry stored in Domino LDAP directory.

WebSphere-Domino SSO scenarios

In scenarios presented in here we have used the following software products:

- ► Windows 2000 with Service Pack 2.
- ► IBM SecureWay Directory Server V3.2.2 for Windows.
- ► IBM DB2 Universal Database V7.2 fp5 for Windows.
- ► IBM HTTP Server 1.3.24 for Windows as the Web server installed on the same machine as the WebSphere server.
- ► Lotus Domino Server R 5.06a.
- ► WebSphere Application Server V5.0
- ► Microsoft Internet Explorer 5.5

There are two different scenarios discussed in this Appendix:

- ► Using SecureWay Directory Server for user registry
- ► Using Domino LDAP for user registry

Using SecureWay Directory Server for user registry

Scenario described in this chapter was based on LDAP schema used also in Tivoli Access Manager examples. Please refer to Chapter 12, "Tivoli Access Manager" on page 367 from more details. Make sure that users defined in LDAP directory are properly mapped into user roles in ITSOBank application. We have tested the scenario with users defined under o=itso suffix. Picture below presents LDAP directory structure used in this scenario.
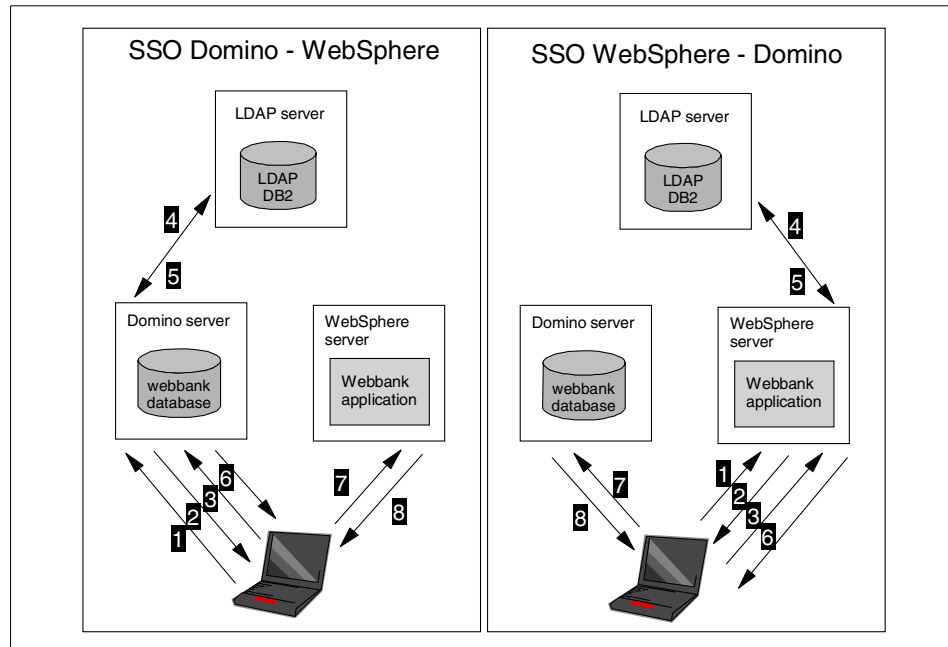
Figure 12-55   Sample SSO Domino - WebSphere using IBM SecureWay Directory

1. A Web user requests a protected resource from the Web server. In case of Domino Server the request is to enter a comment into ITSOBankComments database. In case of a WebSphere user requests a bank transfer.

2. The Web server prompts the user for the authentication information.

3. The user responds to the challenge by supplying the information (user name and password or certificate).

4. The Web Server contacts the LTPA server (Domino or WebSphere) which connects with the IBM SecureWay Directory to verify the authentication information.

5. If the information supplied for the user is correct, the IBM SecureWay Directory responds to the LTPA server with the validated information.

6. The LTPA server uses the returned values to check whether the user has access to the requested resource and issues an LTPA token for the user. The Web server sends the token to the user as an HTTP cookie, which is stored in the user's browser, and serves the requested resource (opening the **ITSOBankComments** database in the case of Domino or **CustomerTransfer.html** in the case of WebSphere).

7. Once the user is authenticated and has the cookie available, the user can request another protected resource to Domino or WebSphere.

8. Domino or WebSphere validates the token provided for the user and tells the Web server to send the requested resource to the browser (as long as the user has access to that resource) without prompting again for user information.

Next sections assume that you have installed and configured WebSphere Application Server with Global Security, LTPA and LDAP user registry enabled. For more information how to do that please refer to Chapter 10, "Administering WebSphere security" on page 233.

Enabling Single Sign-On for WebSphere

In order to configure global security setting for SSO go through the following steps:

1. Configure your WebSphere Application Server to user LDAP user registry

   We assume that WebSphere server is configured to use LDAP user registry. We will not describe detail configuration steps in this chapter. For more information on how to use WebSphere with LDAP user registry please refer to Section 10.13, "Connecting to directory servers (LDAP)" on page 315.

   Our LDAP user registry for WebSphere has been configured with the following attributes:

   Server User ID: `cn=wasadmin,o=ITSO`

   Server User Password: password for wasadmin

   Type: `SecureWay`

   Host: `secsvr01.security.itso.ibm.com`

   Port: `389`

   Base Distinguished Name (DN): `o=ITSO`

   Bind Distinguished Name (DN): `cn=root`

   Bind Password: cn=root user's password

   Search Time-out: `60`

   Other parameters you can leave as default.

2. To enable LTPA for WebSphere follow the steps from Section 10.6.2, "Configuring LTPA for WebSphere" on page 251.

3. To generate the LTPA keys for Single Sign-On follow the steps from Section 10.6.3, "Generate LTPA keys" on page 252.

4. To enable LTPA authentication for WebSphere follow the steps from Section 10.6.4, "Enable LTPA Authentication for WebSphere" on page 253.

Configuring Domino to use IBM SecureWay Directory

To authenticate Web users using the credentials included in the IBM SecureWay Directory, perform the following steps:

1. Create the Directory Assistance Database.

   Domino uses this database to perform searches in other LDAP-compliant directories (such as secondary Domino directories or other LDAP Directory Servers like IBM SecureWay Directory).

   To create a new Directory Assistance database, start the Domino R5 Administration client with a Notes administrator ID, then from the Domino Administrator menu:

2. Choose **File -> Database -> New**. The new database Dialog Box is displayed.

3. Select the server where you want to create the new database. You need to create this database on the server. If You are running Administration Client from your local workstation do not left this field set to Local.

4. Enter a title for the database, for example, **Directory Assistance**.

5. Enter a file name for the database, for example, **da.NSF**.

6. Click the **Template Server...** button and select the Domino server that stores the Directory Assistance template (DA50.NTF); highlight it. Make sure that **Inherit future design changes** is selected, then click **OK**. The New Database window from our scenario is presented on the picture below.
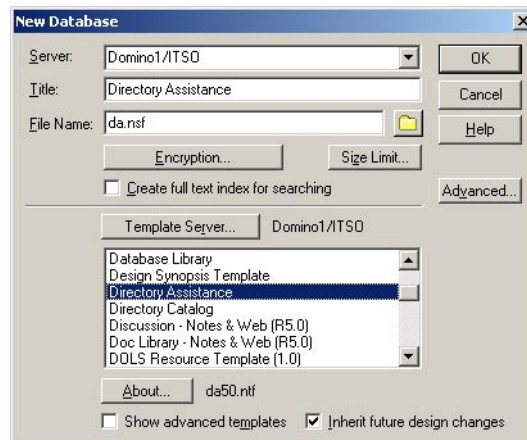
Figure 12-56   Creating Directory Assistance database in Lotus Notes

7. Click **OK** to create a database.

8. If more than one Domino server is going to use that database you will need to replicate it to all these servers.

9. Identify the directory assistance database on the server that will use it. In the Domino Administrator, switch to the **Configuration** tab.

10. In the **Use Directory on** field, choose the server whose Domino Directory you want to modify. In our case it was `Domino1/ITSO`.

11. On the left panel click **Server -> Current Server Document**, this will open document describing your server.

12. Edit the document and on the **Basic** tab enter the file name of your Directory Assistance database into the **Directory Assistance Database Name** field. (If the Directory Assistance database is in a subdirectory, provide the path relative to the data directory, for example: `DIRECTORIES\DA.NSF`).

13. Save and close your Server Document. Server document from scenario presented in this book is shown below.

Figure 12-57   Identify directory assistance database on the server

14. In the same place, on the left pane select **Directory -> Directory Assistance** and click **Add Directory Assistance** button to create a new entry for directory assistance.

15. When directory assistance document is opened specify the following settings on the Basic tab:

    Domain type: `LDAP`

    Domain name: the name of the domain this entry will describe. Domain name must be unique. In our scenario we used `SecureWay`.

Company name: the name of the company associated with this directory, We have used `IBM` in the example.

Search order: specifies the order in which this directory is searched relative to other directories in Directory Assistance database. Because we do not have more directories to search we can leave this field blank.

Group Expansion: select `Yes` to allow Directory Assistance to verify Web user membership when a group is included in a database ACL.

Nested Group Expansion: select `Yes` to allow a lookup of names in groups nested within LDAP directory groups.

Enabled: select `Yes` to enable directory assistance for this entry.

Basic settings from the scenario presented in this book are shown below.

Figure 12-58   Basic settings for Directory Assistance entry

On the Rules tab specify the following settings:

You can specify one or more naming rules that correspond to the hierarchical names of entries in the directory. Directory assistance uses naming rules to determine the order in which to search directories when users provide hierarchical names. For our example, we set the rule */*/*/*/*/* to search for all names in the directory. For more information on Naming Rules and Directory Assistance please refer to Domino R5 Administration Help and Administrator's Guide.

Enable: select `Yes` to enable this specific rule.

Trusted for Credentials: select `Yes` to allow Domino to authenticate only web clients with names that match the rule.

On the LDAP tab specify the following settings:

Hostname: specify DNS Host name for the IBM SecureWay directory, for example `secsvr01.security.itso.ibm.com`

Optional Authentication Credential: the user name and password of the user that Domino server will use when binding to LDAP server. If you do not specify here anything, domino will attempt to bind as anonymous user. In our example we have used `cn=root` user name.

> **Note:** The name and password must correspond to a valid name and password in the directory of the LDAP directory server. If you did not enter a name and password, the Domino server attempts to connect to the LDAP directory server anonymously.
>
> We also recommend using a Notes secret encryption key to encrypt the Directory Assistance document so that only administrators with the encryption key can see the contents of the User name and Password fields; for more details, refer to the Domino R5 Administration Help and Administrator's Guide.

Base DN for search: enter the starting point for LDAP searches. This field is required for SecureWay Directory. In our example we have use `o=itso`.

Perform LDAP searches for: `Notes Clients/Web Authentication`.

Channel encryption: choose `None` to allow Domino server to connect to LDAP without SSL.

Port: `389`

Timeout: `60`

Maximum number of entries returned: `100`.

LDAP tab settings from the scenario tested in this book are presented in the picture below.

Figure 12-59   LDAP settings for directory assistance entry.

16. Save the document.

17. Make sure that the IBM SecureWay Directory is running by checking the list of Windows services on the SecureWay machine and using the TCP/IP `ping` command to test the connection to the SecureWay Directory machine from the Domino Server machine.

18. Restart the Domino server by entering the `restart server` command in the Domino Console.

Enabling Single Sign-On for Domino

Setting up Single Sign-On for the Domino Server involves two main steps:

▶ Creating a Web SSO Configuration Document.

▶ Enabling Single Sign-On.

Follow these steps to enable SSO for Domino:

1. Create a new **Web SSO Configuration Document** in the Domino Directory database.

    This action is only required once (it is only possible to create a Web SSO Configuration document once in your domain) and it should be replicated to all servers participating in the Single Sign-On domain. This document is encrypted for all the participating servers and contains the LTPA keys used to authenticate user credentials.

2. Open the Domino Directory and select **Server -> Servers** to display the view. Click the **Web** action button and select **Create Web SSO Document** in the context menu.

3. A new Document will be displayed with Token Name field set to **LtpaToken**. This name cannot be modified.

4. Enter the DNS domain in the **Token Domain** Field. This value must coincide with the value specified in the **Domain** field in WebSphere when you have configured Single Sign-On panel. This domain name is used when the HTTP cookie is created for Single Sign-On, and determines the scope to which Single Sign-On applies. For our example, we set this domain to **security.itso.ibm.com**.

5. Choose the Domino servers that are going to participate in the SSO scenario (group names are not allowed), for example: `Domino1/ITSO`.

6. In the **Expiration** field enter the maximum number of minutes that the issued token will be valid field. The Default is 30 minutes. In a real life scenario you should match this value with expiration times set on others Single Sign-On participating servers.

7. Click the **Keys**... Action and select **Import WebSphere LTPA Keys** from the drop-down menu.

8. In a dialog box specify the full path name for the WebSphere LTPA keys file exported previously from WebSphere.

9. Click **OK**. A new dialog box will appear, prompting the user for the LTPA password specified when the keys were generated.

10. Enter the password and click **OK** button. When import process completes a confirmation will be displayed.

11. A new WebSphere Information section will appear in the document, as shown in figure below.

Figure 12-60   WebSphere Information in the Web SSO configuration document

The **LDAP realm** is read from the WebSphere Import File and specifies the LDAP host name included in the WebSphere LDAP settings. This name must also coincide with that in the Host Name field specified in the LDAP configuration settings in the Directory Assistance database. When a port is specified in the WebSphere LDAP configuration settings, it will be included in the LTPA key export file in the following format:

`<full dns host name>\:389.`

In our example the line was as follows:

`secsvr01.security.itso.ibm.com\:389.`

But when the LTPA keys are imported in Domino in the **LDAP Realm Name**, the backslash disappears:

`secsvr01.security.itso.ibm.com:389.`

Make sure you add a backslash (\) prior to the colon (:) and replace the value above with the following: `secsvr01.security.itso.ibm.com\:389.`

The LTPA version denotes the version of the WebSphere LTPA implementation. It is read from the LTPA importing file.

12. Click the **Save and Close** button. The document will be saved. To check if the document is present in the Domino Directory, select **Server -> Web Configurations** View and expand the *** - All Servers** - section. The new document created should be displayed as **Web SSO configuration for LtpaToken**.

13. You have now completed the configuration steps for Single Sign-On on Domino. Now we need to tell the server to use this configuration and enable Single Sign-On in Domino server. Open the Server document and select **Ports -> Internet Ports -> Web** tab. Make sure that in the Server document, the TCP/IP port status is enabled and the **Anonymous Authentication** option is set to **No**, as shown below.

Figure 12-61   TCP/IP port status settings for Server Document

14. Configure the Domino HTTP session support by selecting the **Domino Web Engine** tab (**Internet Protocols -> Domino Web Engine**) of the Domino server Document.

15. In the **Session Authentication** field select **Multi-Server**. Selecting this option allows the Web user to log on once in the Domino Server and then gain
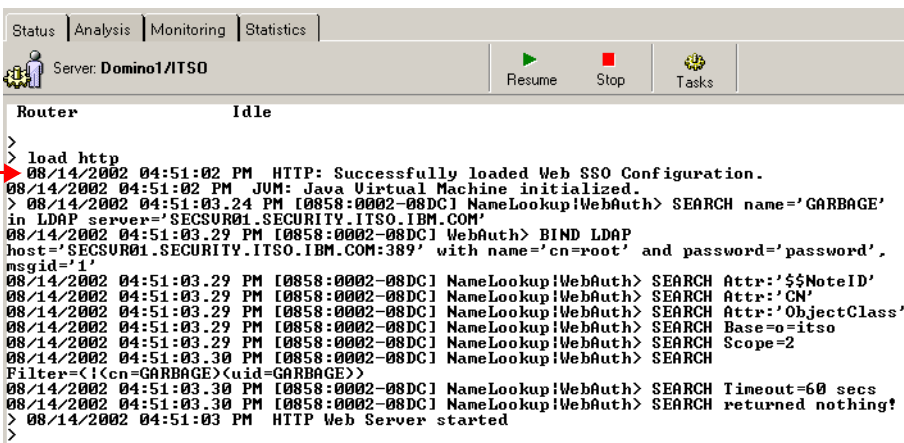
access to another Domino Server or WebSphere server without logging on again.

16. Keeping the Server document open switch to the **Security** tab. Go to the **Web server access** section and select **Fewer name variations with higher security** in the **Web Server Authentication** field.

Selecting this level of restriction makes Domino servers less vulnerable to security attacks by refining how Domino searches for names in the LDAP directory. This option requires users to enter the following user name formats in the user name and password dialog box:

```
Using LDAP Directory for authentication
DN (Full Distinguished Name)
CN (Common Name)
UID or UID with UID= prefix
```

17. Save the Domino Server document.

18. Restart the HTTP server task by entering the `tell http stop` and `load http` commands or the `tell http restart` command in the Domino Console. A new message will appear in the console.

Figure 12-62   Restarting Domino http task with SSO enabled

If a server enabled for SSO cannot find a Web SSO Configuration Document or if it is not included in the Server Names field of the SSO document, so the document cannot be encrypted, the following message should appear in the Domino Server console:

```
HTTP:Error Loading Web SSO configuration.Reverting to single-server session
authentication.
```

This completes configuration of the Domino Server for Single Sign-On with WebSphere. We will now describe what changes you need to provide to ITSOBankComments database in order to test the configuration using ITSOBank application.

Configuring the ITSOBank Domino Application

To test Single Sign-On scenario using ITSOBank application and ITSOBankComments database the Database Access Control List (ACL) needs to be modified by adding the LDAP Users.

1. To add new user names or groups to the ACL, use the LDAP format for the name, but use forward slashes (/) as delimiter rather than commas (,). For example, if the name of a user in the LDAP directory is:

   `cn=clerk01,o=ITSO`

   then you should enter in the database ACL:

   `cn=clerk01/o=ITSO`

2. To add the name of a non-hierarchical LDAP directory group into an ACL, do not include the name of an attribute as part of the entry, but only the value for the attribute. For example, if the name of the LDAP group is: `cn=managergrp,` in the ACL enter only: `managergrp.` However, if the name of the group is hierarchical like: `cn=managergrp,o=ITSO,` then you should enter: `cn=managergrp/o=ITSO.`

   > **Note:** When the LDAP attributes correspond with the attributes used in Notes (for example: `cn,ou,o,c`), the ACL will not display the attributes. For example: `cn=manager01/o=ITSO` appears in the ACL as `manager/ITSO`

3. To add users and groups to the ACL database, make sure that you have manager access to the database and perform the following tasks:

   a. From the Notes Client right-click database icon on the workspace and select **Database -> Access Control**.

   b. This will open access control dialog box.

   c. Set the following ACL for the ITSOBankComments Application database. Table shows only entries related to the application we do not specify here all the entries as presented in the next picture. Similarly we did not change default authorizations to test the scenario. That is why Authorization column is left blank.

Table 12-7   Access control list specified for ITSOBankComments application.

People, servers, groups	User type	Access level	Authorization
Default	Unspecified	NoAccess	None
clerk01/ITSO	Person	Author	
accountant01/ITSO	Person	Author	
manager01/ITSO	Person	Author	
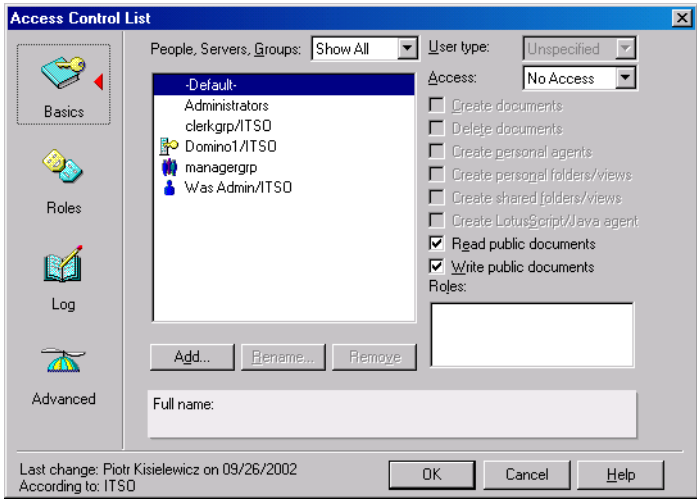
Figure 12-63   ITSOBankComments application ACL

Testing Single Sign-On

We demonstrate the Single Sign-On between Domino and WebSphere in the following scenarios:

► Testing Single Sign-On between Domino and WebSphere

► Testing Single Sign-On between WebSphere and Domino

Testing SSO between Domino and WebSphere

The Web user wants to create a new Comment Document in the ITSOBankComments application database, specifying the database URL in its browser. In our case, this is:

```
http://dominosrv.security.itso.ibm.com/ITSOBank.nsf/Comments?OpenForm
```

As we have defined non anonymous access both to the server and database, Domino will present default server login page, as shown in the picture below.
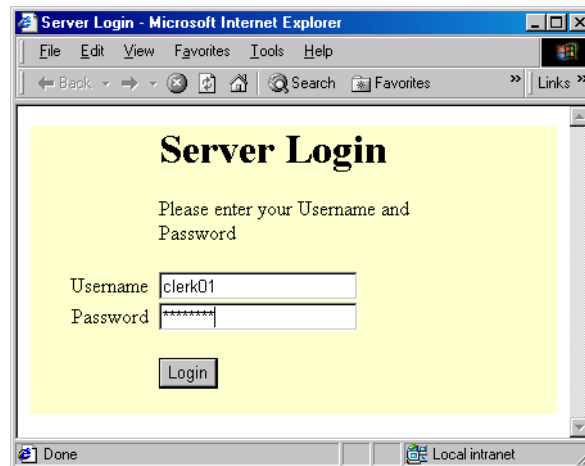

Figure 12-64   Default Domino server login page

When user enters his user name and password and clicks **Login** button, Domino will start to perform searches in LDAP and verifying user and group membership.

After successfully authenticating the user Domino server finds that he has Editor access to the ITSOBank database. It creates LTPA token and sends it to web browser as HTTP cookie. You can view this cookie by typing the following command it the address bar (URL) of the web browser:

```
javascript:alert(document.cookie)
```

The cookie returned by the Web browser is presented on the picture below.

Figure 12-65   LTPA token cookie

As a result of the successful login, user clerk should receive a **Comments** form from ITSOBankComments application.

Figure 12-66   ITSOBank Domino application

Form field **From** should be filled already with the name **clerk01**. After entering subject, comments and pressing **Submit** button, Domino Server will save comments document into a database and present a "Thank you" page to the Web user.

Figure 12-67   Successful submission of comments

By clicking on the link "*Return to our Public web site*" user should be redirected to index.html page running on WebSphere Application Server. Then after selecting Customer Transfer link user clerk should not be prompted for user name and password.

When requesting Internal bank transfer, the user will be presented with "Authorization error" page as the user is not entitled to perform internal transfers.

Testing Single Sign-On between WebSphere and Domino

In this case, the user wants to make a new bank transfer specifying the ITSOBank application URL in its browser. In our case, this is:

```
http://appsrv02.security.itso.ibm.com/itsobank/index.html
```

1. When clicking the link **Customer Transfer** the server prompts the user for authentication information.
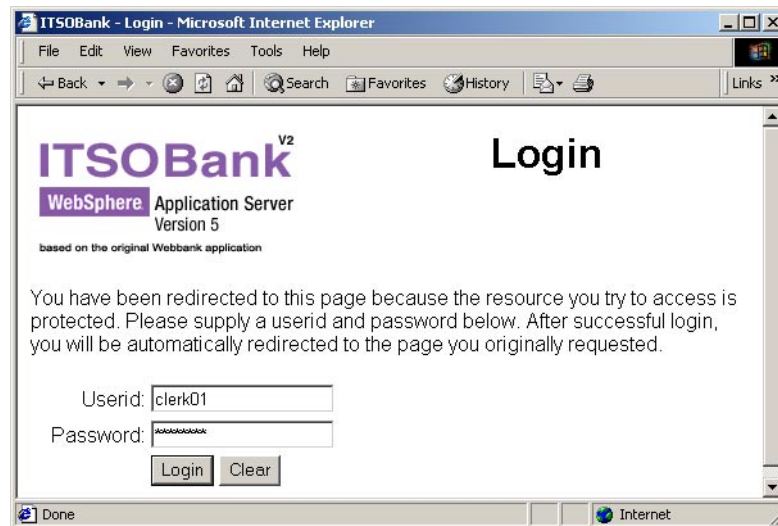
Figure 12-68   ITSOBank login page

2. The user responds to the challenge by supplying the information (user name and password) and clicking the **Login** button.

3. WebSphere will connect with the IBM SecureWay Directory Server to verify the authentication information. If the information supplied is correct, the directory server responds to WebSphere with the valid information.

4. WebSphere uses the returned values to check whether the user has access to the requested resource (customertransfer.html) and issues an LTPA token for the user.

5. The Web server sends the token to the user as an HTTP cookie, then opens the customertransfer.html page

6. At this point, the user can type in the data for the transfer and submit it, go into the domino server to submit comments.

7. When user enters the Comments URL:

   `http://dominosrv.security.itso.ibm.com/ITSOBank.nsf/Comments?OpenForm`

   he should not be presented with the Domino server login page. Domino Opens Comments form with the From field already set to "clerk01"

This completes testing of Single Sign-On between Domino and Websphere when user registry is stored in SecureWay directory.

Using Domino LDAP for user registry

Following diagram presents a scenario for testing Single Sign-On when Domino LDAP server is used a s common user registry.
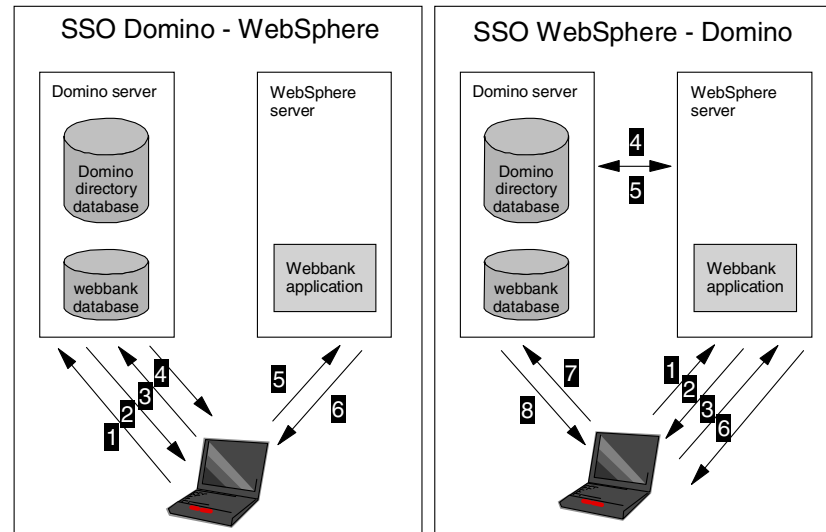
Figure 12-69   Scenario Domino-WebSphere Single Sign-On using Domino LDAP

Log in with WebSphere

The followings steps will describe the Single Sign-On process between Domino and WebSphere, when the user logs into WebSphere first.

1. A Web user submits a request to the Web server (HTTP Server) for a protected resource, to make a new bank transfer.

2. The Web server prompts the user for the authentication information.

3. The user responds by supplying the information (user name and password or certificate).

4. Then the Web server contacts the LTPA server (WebSphere), which connects with the Domino Directory to verify the authentication information.

5. If the information supplied for the user is correct, Domino responds to the WebSphere server with the validated information.

6. The server uses the returned values to check whether the user has access to the requested resource, then issues an LTPA token for the user. The Web server sends the token to the user as an HTTP cookie, which is stored in the

user's browser and serves the requested resource (the CustomerTransfer.html page in case of WebSphere ITSOBank application).

7. Once the user is authenticated and the cookie is available, that user can request another protected resource from Domino or WebSphere.

8. Domino/WebSphere validate the token provided for the user and tell the Web server to send the requested resource to the browser, as long as the user has proper access to that resource, without prompting again with the challenge information.

Log in with Domino

The followings steps will describe the Single Sign-On process between Domino and WebSphere, when the user logs into Domino first.

1. A Web user submits a request to the Web server (Domino) for a protected resource, to create a new Comment document in the ITSOBankComments Application database.

2. Domino prompts the user for the authentication information.

3. The user responds by supplying the information (user name and password or certificate).

4. Domino then verifies the authentication information in the Domino directory, checks whether the user has rights to access to database and issues an LTPA token for the user as an HTTP cookie, which is stored in the user's browser. It then serves the requested resource (it opens the new Comment document).

5. Once the user is authenticated and the cookie is available, that user can request another protected resource from Domino/WebSphere.

6. Domino/WebSphere validate the token provided for the user and tell the Web server to send the requested resource to the browser, as long as the user has proper access to that resource, without prompting again with the challenge information.

The necessary steps to set up Single Sign-On between WebSphere and Domino involves:

► Configuring Websphere to use Domino LDAP

► Enabling Single Sign-On for the WebSphere Application Server

► Enabling Single Sign-On for the Domino Server

Configuring WebSphere LDAP

For the detailed configuration information refer to the Section , "Lotus Domino" on page 458 and follow the instructions from there.

Enabling Single Sign-On for WebSphere

After configuring WebSphere Application Server to use Domino LDAP, SSO configuration is identical to the one discussed in previous section. Please refer to , "Enabling Single Sign-On for WebSphere" on page 490 for details. Remember that you should generate LTPA keys always after successful configuration of LDAP user registry.

> **Important:** Do not forget to enter domain name in the Single sign-on section of the LTPA configuration panel.

Enabling Single Sign-On for Domino

When using Domino directory as a user registry, Domino Server does not need to use directory assistance as described in the previous section. After putting all the application users and groups into your Domino directory, you can follow the instruction from section , "Enabling Single Sign-On for Domino" on page 495 for importing LTPA keys and enabling Single Sign-On for the Domino Server.

For our example scenario we have defined the following users and groups in Domino Directory:

Table 12-8   Users and groups defined in domino directory for ITSO application

Group name	Group members
managergrp/ITSO	manager01/ITSO
clerkgrp/ITSO	clerk01/ITSO
accountantgrp/ITSO	accountant01/ITSO
consultantgrp/ITSO	consultant01/ITSO

We have mapped Domino directory groups to a corresponding user roles in ITSO bank application, and accordingly modified ACL in ITSOBankCopmments application database.

If your server is already configured to use Single Sign-On, please remember that reconfiguration does not mean creating new Web Single Sign-On Configuration document. On a server it may only be one Web Single Sign-On Configuration Document. So, if you have one already, you should edit it and import new LTPA keys.

Testing Single Sign-On

For testing this scenario you can follow testing instructions from previosu section. Please refer to , "Testing Single Sign-On" on page 500.

D

Using wsadmin scripting for security configuration

This appendix gives a short introduction of WebSphere Application Server's new administrative scripting language and then provides some sample scripts.

The sample scripts provided here come handy when security changes need to be performed on the system.

Please note that the scripts are very basic and not very intelligent, they will only work in a single server environment and some information might be hard-coded in the source.

With little modification you can change the scripts to take parameters from the command line and perform modification according to the given parameters; or change them to make it work in a cell not only on one application server.

wsadmin scripting

WebSphere Application Server 5.0 provides new administration scripting engine that can be used to manage and deploy applications. Scripts are executed under new tool wsadmin. The wsadmin client uses the Bean Scripting Framework (BSF), and is based on JMX.

This appendix introduces scripting language and presents few simple examples that can be customized for different purpose.

In WebSphere Application Server V4.0 wscp commands were used for both configuration queries and updates, and operational commands. In V5.0, a distinction is made between configurational and operational commands.

► Configuration functions - deal WebSphere Application Server V5.0 installation and configuration

► Operational functions - deal with management of currently running objects in WebSphere Application Server V5.0 installations.

Scripts deal with both categories of objects. For example, an application server is divided into two distinct entities. One entity represents the configuration of the server, which resides persistently in a repository on permanent storage. You can create, query, change, or remove this configuration without starting an application server process. The second entity represents the running instance of an application server by a Java Management Extensions (JMX) MBean. This instance can have attributes that you can interrogate and change, and operations that you can invoke. These operational actions taken against a running application server do not have an effect on the persistent configuration of the server. The attributes that support manipulation from an MBean differ from the attributes the corresponding configuration supports. The configuration can include many attributes that you cannot query or set from the live running object.

When you run wsadmin in an environment where global security is enabled, you will need to supply authentication information in order to communicate with server. The user's name and password may be specified either in command line arguments for wsadmin or in sas.client.props file. Changes introduced into the properties file will depend on whether RMI or SOAP connector is used to communicate with the server. Remember that if you specify user's name and password in command line, it will override these information in properties file.

JACL language that is used in scripts allows to create procedures. Procedures may be grouped into a profile files that can be passed to wsadmin in a command line to create a custom environment for wsadmin execution. These procedures can then be used as normal JACL commands. For more information about how to create profiles please refer to WebSphere Application Server info center.

Preparing and testing the wsadmin client

The wsadmin scripting tool can run with a command prompt where the user can feed the commands to the interpreter and execute tem from a console; or wsadmin can run in a silent mode where users can pass a script file as a parameter to the tool and execute a whole sequence of commands.

The scripts provided below can be saved in files and feed then to the wsadmin tool or you can also start wsadmin as a command line interpreter and run the commands line-by-line from the scripts.

Since the wsadmin tool is basically a Java application accessing the WebSphere Application Server, when global security is enabled the application has to provide the right user name and password to be able to run the scripts.

There are different ways to pass the username and passwod, in the command line or modifying the sas.client.props file.

Running the script with command line parameters

Issue the `wsadmin` command with the following parameters:

▶ -username <administrator_user_name>

▶ -password <administrator_password>

For example:

```
wsadmin -username cn=wasadmin,o=itso -password password myscript.jacl
```

Editing the sas.client.props file

Follow the steps to provide the user name and password in the properties file for the client.

1. Open *sas.client.props* file in your text file editor, The file is located in <WAS_INSTALL>/properties directory

2. Go to the section Authentication Configuration and find # JMX SOAP connector identity

3. In the line *com.ibm.SOAP.loginUserid=* write your security server user ID (the same that you have entered while configuring Global Security. If you are using LDAP distinguished names do not put names in quotes, just write as they are defined in LDAP server. For our test we have used LDAP user `cn=wasadmin,o=itso`.

4. In the line *com.ibm.SOAP.loginPassword=* put the password for that user.

5. save the file and run wsadmin client without any parameters, You should get the output similar to the presented below.

Example 12-9   Example output after successful connection from wsadmin client

```
C:\>wsadmin

WASX7209I: Connected to process "server1" on node mka0klfc using SOAP
connector;
  The type of process is: UnManagedProcess
WASX7029I: For help, enter: "$Help help"
wsadmin>
```

Now you are ready to start writing scripts.

Sample scripts

This section lists some useful scripts to make the configuration more convenient for WebSphere.

Each script will show and teach something new with the scripting language, and at the end you will learn how to create your own scripts based on these. Also you can always refer to the InfoCenter for scripting references and more scripting samples.

Setting global security

The following script can enable or disable global security for the application server depending on the parameter value passed to the interpreter. For more information on this script check the comments in teh source below.

Example 12-10   globalsecurity.jacl

```
# global security
# usage: wsadmin globalsecurity.jacl [enable | disable]
# store the pointer to the security object
set security_item [$AdminConfig list Security]
# store the parameter from the command line
set argv0 [lindex $argv 0]
# initialize the value variable
set value null
# checking the parameter passed to the script
# setting the value according to the parameter
if {[regexp $argv0 enable]} { set value true }
if {[regexp $argv0 disable]} { set value false }
if {[regexp $value null]} {
 puts "Wrong parameter, use enable / disable"
 return
}
# modifying the attribute in the configuration
```

```
$AdminConfig modify $security_item [list [list enabled $value]]
# saving the configuration
$AdminConfig save
# this will dump the current security settings for testing
$AdminConfig show $security_item
```

As you see, the script assumes that only one application server is defined for the node since it expects only one secuirty object returning from the query.

Configuring user registry

The following script wil change the user registry to LDAP.

Example 12-11   changeUR.jacl

```
# change user registry
# the type of the user registry is hardcoded to LDAP
# it also assumes that there is an entry already with the name of LDAP defined
for the security object
set user_regName "LDAP"
# get the security object
set security_item [$AdminConfig list Security]
# list all the user registries defined
set user_regs [$AdminConfig list UserRegistry]
# find the one that starts with the name we set at the beginning of the script
foreach user_reg $user_regs { if {[regexp $user_regName $user_reg]} { set
new_user_reg $user_reg; break }}
# modify the user registry attribute for the security object
$AdminConfig modify $security_item [list [list activeUserRegistry
$new_user_reg]]
# saving the configuration
$AdminConfig save
```

You can also set the user registry to local OS or custom use registry. WebSphere Application Server has all three user registry entries predefined you just have to provide the right name for the script in line #5. You can list the user registry objects by running the following command:

```
$AdminConfig list UserRegistry
```

The result should look like the following example:

```
(cells/appsrv01Node:security.xml#CustomUserRegistry_1)
(cells/appsrv01Node:security.xml#LDAPUserRegistry_1)
(cells/appsrv01Node:security.xml#LocalOSUserRegistry)
```

Then use the name of one of the listed objects: LocalOSUserRegistry, LDAPUserRegistry_1, CustomUserRegistry_1.

Since the condition is not doing an exact match it is enough to provide only part of the name, like: Local, LDAP, Custom.

Creating a new SSL entry

The following script will create a new SSL entry. Note that the file names and passwords are hard-coded in the file, you have to modify the values or change the script to take parameters.

Example 12-12   addSSLentry.jacl

```
# new SSL entry in the SSL repertoire
# setting the security object
set security_root [$AdminConfig list Security]
# setting the variables for the entry
set ssl_alias "new SSL entry"
set ssl_clientAuthentication [list clientAuthentication false]
set ssl_enableCryptoHardwareSupport [list enableCryptoHardwareSupport false]
set ssl_keyFileFormat [list keyFileFormat "JKS"]
set ssl_keyFileName [list keyFileName "c:\\was\\etc\\clientkeyfile.jks"]
set ssl_keyFilePassword [list keyFilePassword "password"]
set ssl_securityLevel [list securityLevel "HIGH"]
set ssl_trustFileFormat [list trustFileFormat "JKS"]
set ssl_trustFileName [list trustFileName "c:\\was\\etc\\trustkeyfile.jks"]
set ssl_trustFilePassword [list trustFilePassword "password"]
# this long line puts the attributes for the object together from the variables
and values
set ssl_def [list $ssl_clientAuthentication $ssl_enableCryptoHardwareSupport
$ssl_keyFileFormat $ssl_keyFileName $ssl_keyFilePassword $ssl_securityLevel
$ssl_trustFileFormat $ssl_trustFileName $ssl_trustFilePassword]
# defining the whole SSL object
set ssl_entry [list [list alias $ssl_alias] [list setting $ssl_def] ]
# creating the new entry
$AdminConfig create SSLConfig $security_root $ssl_entry repertoire
# saving the configuration
$AdminConfig save
```

Creating a J2C Authentication Entry

The following script is very similar to the previous one, Creating a new SSL entry. This one will create a new J2C authentication entry for WebSphere.

Example 12-13   newJ2Centry.jacl

```
# create a new J2C authentication entry
# set the security object
set security_root [$AdminConfig list Security]
# set the attributes for the new object
set auth_alias [list alias "itsobankds_auth"]
```

```
set auth_descr [list description "ITSOBank DataSource authentication alias"]
set auth_userId [list userId "dbuser01"]
set auth_password [list password "password"]
# put the new object together
set auth_entry [list $auth_alias $auth_descr $auth_userId $auth_password]
# create the new object
$AdminConfig create JAASAuthData $security_root $auth_entry
# saving the configuration
$AdminConfig save
```

Assign a J2C entry to a DataSource

Once you have at least one J2C authentication entry in your system you can assign them to certain objects like: DataSource.

Example 12-14   assignJ2Centry.jacl

```
# change authentication alias for DataSource
# usage: wsadmin assignJ2Centry.jacl datasource_name alias_name
# storing the parameters
set ds_arg [lindex argv 0]
set alias_arg [lindex argv 1]
# finding the datasource based on name
set datasources [$AdminConfig list DataSource]
foreach datasource $datasources {if {[regexp $ds_arg $datasource]} { set
datasource_root $datasource; break }}
# modify the authentication alias attribute for the datasource
$AdminConfig modify $datasource_root [list [list authDataAlias $alias_arg]]
# saving the configuration
$AdminConfig save
```

E

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this 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/`SG24????

Alternatively, you can go to the IBM Redbooks Web site at:

    ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG24????.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name   *Description*

          SG246573.zip   contains the ITSOBank sample application with security enhancements.

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space: 10 MB minimum
Operating System: Windows 2000 with Service Pack 3 or AIX 5L with Maintenance Level 1
Processor: 500MHz or higher
Memory: 512MB or more

How to use the Web material

Create a subdirectory (folder) on your workstation, for example: C:\temp\sg246573, and unzip the contents of the Web material zip file into this folder.

Refer to Appendix A, "Sample application" on page 441 for instructions on how to deploy the sample application in the runtime environment.

The book itself will refer to the sample application at several places, and almost everywhere when some kind of sample is shown.

Abbreviations and acronyms

AAT	Application Assembly Tool	**J2EE**	Java 2 Enterprise Edition
AC	Administrator's Console	**JCE**	Java Cryptography Extension
ACL	Access Control List	**JDBC**	Java Database Connectivity
AE	Advanced Edition	**JKS**	Java Key Store
API	Application Programmer's Interface	**JSP**	JavaServer Page
		JVM	Java Virtual Machine
CA	Certificate Authority	**KDB**	Key Database, file extension for IBM's iKeyman
CN	Common Name		
CRL	Certificate Revocation List	**LDAP**	Lightweight Directory Access Protocol
CSI	Common Secure Interoperability	**LDIF**	LDAP Data Interchange Format
CSR	Certificate Signing Request	**LTPA**	Lightweight Third Party Authentication
DD	Deployment Descriptor		
DMZ	De-Militarized Zone	**ORB**	Object Request Broker
DN	Distinguished Name	**OS**	Operating System
DNS	Domain Name Server	**OU**	Organizational Unit
EAR	Enterprise Application Archive	**PD**	Policy Director
		PID	Process ID
EJB	Enterprise Java Bean	**PKI**	Public Key Infrastructure
GSO	Global Sign-On	**QOP**	Quality Of Protection
GUI	Graphical User Interface	**RA**	Registration Authority
HTML	Hypertext Markup Language	**RFC**	Request For Comments
HTTP	Hypertext Transfer Protocol	**RMI**	Remote Method Invocation
IBM	International Business Machines Corporation	**RPC**	Remote Procedure Call
IDE	Integrated Development Environment	**RPSS**	Reverse Proxy Security Server
IHS	IBM HTTP Server	**SAS**	Secure Association Service (IBM propriatery)
IIOP	Internet Inter ORB Protocol		
IOR	Interoperable Object Reference	**SAS**	Security Attribute Service (OMG CSI)
IT	Information Technology	**SOA**	Service Oriented Architecture
ITSO	International Technical Support Organization	**SOAP**	Simple Object Access Protocol

SPI	Service Provider Interfaces
SSL	Secure Socket Layer
SSO	Single Sign-On
SWAM	Simple WebSphere Authentication Mechanism
UDDI	Universal Description, Discovery, Integration
URI	Unified Resource Identifier
URL	Unified Resource Locator
VPN	Virtual Private Network
WAR	Web Application Archive
WAS	WebSphere Application Server
WSAD	WebSphere Studio Application Developer
WSCP	WebSphere Control Program
WSDL	Web Services Description Language
XML	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 redbook.

IBM Redbooks

For information on ordering these publications, see "How to get IBM Redbooks" on page 521.

► *????full title???????,* xxxx-xxxx

► *????full title???????,* xxxx-xxxx

► *????full title???????,* xxxx-xxxx

Other resources

These publications are also relevant as further information sources:

► *????full title???????,* xxxx-xxxx

► *????full title???????,* xxxx-xxxx

► *????full title???????,* xxxx-xxxx

Referenced Web sites

These Web sites are also relevant as further information sources:

► Description1

   `http://????????.???.???/`

► The Java 2 Platform Specification v1.3 at

   `http://java.sun.com/j2ee/docs.html`

How to get IBM Redbooks

You can order hardcopy Redbooks, as well as view, download, or search for Redbooks at the following Web site:

   ibm.com/redbooks

You can also download additional materials (code samples or diskette/CD-ROM images) from that site.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Index

IBM

Redbooks

IBM WebSphere V5.0 Security Handbook WebSphere Handbook series

(1.5" spine)
1.5"<-> 1.998"
789 <->1051 pages

IBM

Redbooks

IBM WebSphere V5.0 Security Handbook

(1.0" spine)
0.875"<->1.498"
460 <-> 788 pages

IBM WebSphere V5.0 Security Handbook WebSphere

(0.5" spine)
0.475"<->0.875"
250 <-> 459 pages

IBM

Redbooks

IBM WebSphere V5.0 Security Handbook WebSphere Handbook

(0.2"spine)
0.17"<->0.473"
90<->249 pages

IBM

Redbooks

(0.1"spine)
0.1"<->0.169"
53<->89 pages

IBM

Redbooks

IBM WebSphere V5.0 Security Handbook

IBM

Redbooks

IBM WebSphere V5.0 Security Handbook

(2.0" spine)
2.0" <-> 2.498"
1052 <-> 1314 pages

(2.5" spine)
2.5"<->nnn.n"
1315<-> nnnn pages

IBM ®

IBM WebSphere V5.0 Security Handbook
WebSphere Handbook series

Redbooks

WebSphere Application Server security in details

End-to-end security design with Patterns for e-business

Security integration with Tivoli Access Manager

This IBM Redbook provides IT Architects, IT Specialists, application designers, application developers, application assemblers, application deployers and consultants with informaiton to design, develop and deploy secure e-business applications using WebSphere Application Server V5.

The WebSphere security part provides a detailed overview of WebSphere Application Server V5 Security. It starts up with J2EE security, goes into details with the modules and components of a J2EE enterprise application; you can also read about programmatic security techniques. The last chapter of this part shows all the security related administrtive items in WebSphere Application Server V5.

The End-to-end security part offers details about end-to-end security solutions where WebSphere Application Server V5 is part of an enterprise solution. You will find an introduction of Patterns for e-business, in which security is in focus. A very important chapter of this part will discuss the integration between WebSphere Application Server V5 and Tivoli Access Manager.

Finally the Appendixes provide additional information related to chapters from the previous two parts and also describes the sample application available together with the book.