

Orbix Mainframe 6.3.1

Security Guide

Micro Focus
The Lawn
22-30 Old Bath Road
Newbury, Berkshire RG14 1QN
UK
<https://www.microfocus.com>

© Copyright 2021 Micro Focus or one of its affiliates.

MICRO FOCUS, the Micro Focus logo and Orbix are trademarks or registered trademarks of Micro Focus or one of its affiliates.

All other marks are the property of their respective owners.

2021-03-18

Contents

List of Tables	ix
List of Figures	xi
Preface	xiii

Part 1 Introducing Security

Chapter 1 Orbix Security Framework	1
Introduction to the iSF	2
iSF Features	3
Example of an iSF System	4
Security Standards	6
Orbix Security Service	7
Orbix Security Service Architecture	8
iSF Server Development Kit	9
Secure Applications	10
ART Security Plug-Ins	11
Secure CORBA Applications	13
Administering the iSF	15
Overview of iSF Administration	16
Secure ASP Services	18

Part 2 Orbix Security Framework Administration

Chapter 2 Transport Layer Security	21
What does Orbix Provide?	22
How TLS Provides Security	24
Authentication in TLS	25

Certificates in TLS Authentication	27
Privacy of TLS Communications	28
Integrity of TLS Communications	29
Chapter 3 Securing Applications and Services	31
Connecting to an Off-Host iS2 Server	32
Securing CORBA Applications	33
Overview of CORBA Security	34
Securing Communications with SSL/TLS	36
Specifying Fixed Ports for SSL/TLS Connections	46
Securing Two-Tier CORBA Systems with iSF	48
Securing Three-Tier CORBA Systems with iSF	53
Securing Orbix Services	59
Caching of Credentials	60
Chapter 4 Managing Access Control Lists	61
CORBA ACLs	62
Overview of CORBA ACL Files	63
CORBA Action-Role Mapping ACL	64
Centralized ACL	68
Local ACL Scenario	69
Centralized ACL Scenario	71
Customizing Access Control Locally	77
Chapter 5 Managing Certificates	79
What are X.509 Certificates?	80
Certification Authorities	82
Commercial Certification Authorities	83
Private Certification Authorities	84
Certificate Chaining	85
PKCS#12 Files	87
Managing Certificates on z/OS	88
Importing Certificates from Another Platform into RACF	89
Creating Certificates for an Application Using RACF	94
Specifying the Source of Certificates for an z/OS Application	95

Part 3 SSL/TLS Administration

Chapter 6	Choosing an SSL/TLS Toolkit	99
	Toolkit Replaceability	100
	System SSL Toolkit on z/OS	101
Chapter 7	Configuring SSL/TLS Secure Associations	103
	Overview of Secure Associations	104
	Setting Association Options	106
	Secure Invocation Policies	107
	Association Options	108
	Choosing Client Behavior	110
	Choosing Target Behavior	112
	Hints for Setting Association Options	114
	Specifying Cipher Suites	119
	Supported Cipher Suites	120
	Setting the Mechanism Policy	126
Chapter 8	Configuring SSL/TLS Authentication	131
	Requiring Authentication	132
	Target Authentication Only	133
	Target and Client Authentication	136
	Specifying an Application's Own Certificate	139
	Advanced Configuration Options	142
	Setting a Maximum Certificate Chain Length	143
	Applying Constraints to Certificates	144

Part 4 CSlv2 Administration

Chapter 9	Introduction to CSlv2	149
	CSlv2 Features	150
	Basic CSlv2 Scenarios	152
	CSlv2 Authentication over Transport Scenario	153
	CSlv2 Identity Assertion Scenario	154

Chapter 10	Configuring CSlv2 Authentication over Transport	157
	CSlv2 Authentication Scenario	158
	SSL/TLS Prerequisites	162
	Requiring CSlv2 Authentication	164
	Providing an Authentication Service	167
	Providing a Username and Password	168
	Sample Configuration	172
	Sample Client Configuration	173
	Sample Server Configuration	175
Chapter 11	Configuring CSlv2 Identity Assertion	177
	CSlv2 Identity Assertion Scenario	178
	SSL/TLS Prerequisites	182
	Enabling CSlv2 Identity Assertion	184
	Sample Configuration	186
	Sample Client Configuration	187
	Sample Intermediate Server Configuration	189
	Sample Target Server Configuration	191
Part 5	CORBA Security Programming	
Chapter 12	Programming Policies	195
	Setting Policies	196
	Programmable SSL/TLS Policies	199
	Introduction to SSL/TLS Policies	200
	The QOPPolicy	202
	The EstablishTrustPolicy	203
	The InvocationCredentialsPolicy	204
	Interaction between Policies	205
	Programmable CSlv2 Policies	206
Chapter 13	Authentication	209
	Using the Principal Authenticator	210
	Introduction to the Principal Authenticator	211
	Creating SSL/TLS Credentials	214
	Creating CSlv2 Credentials	217

Using a Credentials Object	221
Retrieving Own Credentials	223
Retrieving Own Credentials from the Security Manager	224
Parsing SSL/TLS Own Credentials	225
Retrieving Target Credentials	226
Retrieving Target Credentials from an Object Reference	227
Parsing SSL/TLS Target Credentials	229
Retrieving Received Credentials	230
Retrieving Received Credentials from the Current Object	231
Parsing SSL/TLS Received Credentials	232
Chapter 14 Validating Certificates	233
Overview of Certificate Validation	234
The Contents of an X.509 Certificate	237
Parsing an X.509 Certificate	238
Controlling Certificate Validation	239
Certificate Constraints Policy	240
Certificate Validation Policy	243
Obtaining an X.509 Certificate	247

Part 6 Appendices

Appendix A Security Configuration	251
Applying Constraints to Certificates	253
initial_references	255
plugins:atli2_tls	256
plugins:csi	257
plugins:gsp	258
plugins:https	263
plugins:iiop_tls	264
plugins:security	269
plugins:systemssl_toolkit	270
policies	272
policies:csi	279
policies:https	282
policies:iiop_tls	289
principal_sponsor	300

principal_sponsor:csi	304
Appendix B ASN.1 and Distinguished Names	307
ASN.1	308
Distinguished Names	309
Appendix C Association Options	313
Association Option Semantics	314
Appendix D SSL/TLS Sample Configurations	317
SSL/TLS Sample Configurations on z/OS	318
Appendix E Security Recommendations	323
General Recommendations	324
Orbix Services	325
Appendix F Action-Role Mapping DTD	327
Index	331

List of Tables

Table 1: Terminology Describing Secure Client Sample Configurations	36
Table 2: Terminology Describing Secure Server Sample Configurations	38
Table 3: Description of Different Types of Association Option	115
Table 4: Setting EstablishTrustInTarget and EstablishTrustInClient Association Options	116
Table 5: Setting Quality of Protection Association Options	116
Table 6: Setting the NoProtection Association Option	118
Table 7: Cipher Suite Definitions	122
Table 8: Policy Management Objects	196
Table 9: Commonly Used Attribute Types	310
Table 10: AssociationOptions for Client and Target	314

LIST OF TABLES

List of Figures

Figure 1: Example System with a Standalone Orbix Security Service	4
Figure 2: Security Plug-Ins in a CORBA Application	13
Figure 3: A Secure CORBA Application within the iSF	34
Figure 4: Two-Tier CORBA System in the iSF	48
Figure 5: Three-Tier CORBA System in the iSF	53
Figure 6: Local ACL Scenario	69
Figure 7: Centralized ACL scenario	71
Figure 8: Custom ClientAccessDecision in an Orbix Application	77
Figure 9: A Certificate Chain of Depth 2	85
Figure 10: A Certificate Chain of Depth 3	86
Figure 11: Configuration of a Secure Association	105
Figure 12: Target Authentication Only	133
Figure 13: Target and Client Authentication	136
Figure 14: Elements in a PKCS#12 File	140
Figure 15: Basic CSIV2 Authentication over Transport Scenario	153
Figure 16: Basic CSIV2 Identity Assertion Scenario	154
Figure 17: CSIV2 Authentication Over Transport Scenario	159
Figure 18: Java Dialog Window for GSSUP Username and Password	169
Figure 19: CSIV2 Identity Assertion Scenario	179
Figure 20: Validating a Certificate	234
Figure 21: Using a CertValidator Callback	236

LIST OF FIGURES

Preface

What is Covered in this Book

This book is a guide to administering and programming secure CORBA applications with Orbix.

The Orbix security framework (ISF) provides the underlying security infrastructure for performing authentication and authorization.

Who Should Read this Book

This guide is intended for the following audience:

- Security administrators.
- CORBA C++ developers.
- CORBA Java developers.

A prior knowledge of CORBA is assumed.

Organization of this guide

This guide is divided into the following parts:

Part I “[Introducing Security](#)”

This part describes how TLS provides security, and how Orbix works.

Part II “[Orbix Security Framework Administration](#)”

This part describes how to administer the Orbix Security Framework.

Part III “[SSL/TLS Administration](#)”

This part explains how to configure and manage Orbix in detail.

Part IV “CSlv2 Administration”

This part explains how to configure and manage CSLv2 in detail.

Part V “CORBA Security Programming”

This part explains how to program the SSL/TLS and CSLv2 APIs in your security-aware CORBA applications.

Appendices

The appendices list further technical details.

Related Documentation

The *Orbix Programmer's Guide C++ Edition* and *Orbix Programmer's Reference C++ Edition* provide details about developing Orbix applications in C++ in various environments, including z/OS.

The latest updates to the Orbix documentation can be found at:

<https://www.microfocus.com/documentation/orbix/>

Additional Resources

The Knowledge Base contains helpful articles, written by experts, about Orbix Mainframe, and other products:

<https://community.microfocus.com/t5/Orbix/ct-p/Orbix>

If you need help with Orbix Mainframe or any other products, contact technical support:

<https://www.microfocus.com/en-us/support/>

Typographical Conventions

This book uses the following typographical conventions:

Constant width Constant width (courier font) in normal text represents portions of code and literal names of items such as classes, functions, variables, and data structures. For example, text might refer to the `CORBA::Object` class.

Constant width paragraphs represent code examples or information a system displays on the screen. For example:

```
#include <stdio.h>
```

Italic Italic words in normal text represent *emphasis* and *new terms*.

Italic words or characters in code and commands represent variable values you must supply, such as arguments to commands or path names for your particular system. For example:

```
% cd /users/your_name
```

Note: Some command examples may use angle brackets to represent variable values you must supply. This is an older convention that is replaced with *italic* words or characters.

Keying Conventions

This book uses the following keying conventions:

No prompt	When a command's format is the same for multiple platforms, a prompt is not used.
%	A percent sign represents the UNIX command shell prompt for a command that does not require root privileges.
#	A number sign represents the UNIX command shell prompt for a command that requires root privileges.
>	The notation > represents the Windows command prompt.
...	Horizontal or vertical ellipses in format and syntax descriptions indicate that material has been eliminated to simplify a discussion.
[]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.
	A vertical bar separates items in a list of choices enclosed in {} (braces) in format and syntax descriptions.

Part 1

Introducing Security

In this part

This part contains the following chapters:

Orbix Security Framework	page 1
Transport Layer Security	page 21

Orbix Security Framework

The Orbix Security Framework provides the common underlying security framework for all types of applications in Orbix, including CORBA and Web services applications. This chapter provides an introduction to the main features of the iSF.

In this chapter

This chapter discusses the following topics:

Introduction to the iSF	page 2
Orbix Security Service	page 7
Secure Applications	page 10
Administering the iSF	page 15

Introduction to the iSF

Overview

This section provides a brief overview of and introduction to the Orbix Security Framework, which provides a common security framework for all components of Orbix.

In this section

This section contains the following subsections:

iSF Features	page 3
Example of an iSF System	page 4
Security Standards	page 6

iSF Features

Overview

The Orbix Security Framework is a scalable, standards-based security framework with the following features:

- Pluggable integration with third-party enterprise security systems.
- Out-of-the-box integration with flat file, or LDAP security systems.
- Centralized management of user accounts.
- Role-Based Access Control.
- Role-to-permission mapping supported by access control lists.
- Unified security platform works across CORBA and Web services.
- Security platform is ART-based.
- Logging.

Example of an iSF System

Overview

Figure 1 shows an example of an iSF system that features a standalone Orbix security service (iS2 server), which can service remote requests for security-related functions.

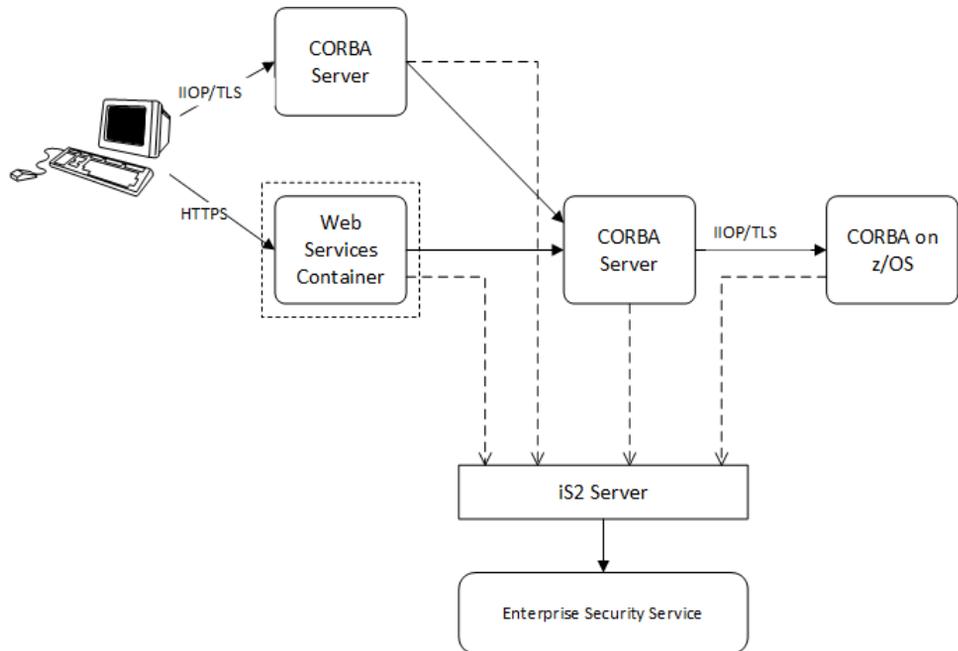


Figure 1: Example System with a Standalone Orbix Security Service

Orbix security service

The Orbix security service is the central component of the Orbix Security Framework, providing an authentication service, an authorization service and a repository of user information and credentials. When the Orbix security service is deployed in standalone mode, all kinds of applications, including CORBA applications and Web services, can call it remotely.

Enterprise security service

The Orbix security service is designed to integrate with a third-party enterprise security service (ESS), which acts as the primary repository for user information and credentials. Integration with an ESS is supported by a variety of *iSF adapters*. The following adapters are currently supported by iSF:

- LDAP adapter.

The following adapter is provided for use in simple demonstrations (but is *not* supported in production environments):

- File adapter.

In addition, it is possible to build your own adapters using the iSF Adapter SDK—see [“iSF Server Development Kit” on page 9](#).

Propagating security credentials

The example in [Figure 1 on page 4](#) assumes that a user’s credentials can be propagated from one application to another. There are fundamentally two different layers that can propagate security credentials between processes in an iSF distributed system:

- [Transport layer](#).
 - [Application layer](#).
-

Transport layer

Security at the transport layer enables security information to be exchanged during the security handshake, which happens while the connection is being established. For example, the SSL/TLS standard enables X.509 certificates to be exchanged between a client and a server during a security handshake.

Application layer

Security at the application layer enables security information to be propagated *after* connection establishment, using a protocol layered above the transport. For example, the CORBA common secure interoperability v2.0 (CSiv2) protocol propagates security information by embedding security data in IIOP messages, which are layered above TCP/IP.

The CSiv2 protocol can be used to propagate any of the following kinds of credential:

- Username/password/domain.
- Username only.
- Single-sign on (SSO) token.

Security Standards

Overview

One of the goals of the iSF is to base the security framework on established security standards, thereby maximizing the ability of iSF to integrate and interoperate with other secure systems. This section lists the security standards currently supported by the iSF.

Standards supported by iSF

The following security standards are supported by iSF:

- HTTP login mechanisms—that is, HTTP basic authentication and HTTP form-based authentication.
- Secure Sockets Layer / Transport Layer Security (SSL/TLS), from the Internet Engineering Task Force, which provides data security for applications that communicate across networks.
- CCITT X.509, which governs the form of security certificates based on public (asymmetric) key systems)
- OMG Common Secure Interoperability specification (CSIv2)
- The XML Key management Specification (XKMS), which specifies the protocols for distributing and registering public keys. XKMS is composed of the XML Key Information Service Specification (X-KISS), and the XML Key Registration Service Specification (X-KRSS). XKMS provides the Public Key Infrastructure (PKI) support in iSF.
- Security Assertion Markup Language (SAML) from the Organization for the Advancement of Structured Information Standards (OASIS), which is the XML security standard for exchanging authentication and authorization information. The SAML specification provides bindings for various transport protocols including HTTP/HTTPS and SOAP.
- Secure Multipurpose Internet Mail Extensions (S/MIME), which is a specification for secure electronic mail, and is designed to add security to e-mail messages in MIME format.
- WS-Security, which a proposed standard from Microsoft, IBM, and VeriSign. It defines a standard set of SOAP extensions, or message headers, that can be used to implement integrity and confidentiality in Web services applications.
- Java Authentication and Authorization Service (JAAS)

Orbix Security Service

Overview

The Orbix security service is the central component of the Orbix Security Framework. This section provides an overview of the main Orbix security service features.

In this section

This section contains the following subsections:

Orbix Security Service Architecture	page 8
iSF Server Development Kit	page 9

Orbix Security Service Architecture

iSF client API

The GSP plug-in accesses the Orbix security service through the iSF client API, which is a private Orbix-proprietary API. This API exposes general security operations, such as authenticating a username and password, retrieving a user's roles, and so on. Two language versions of the iSF client API are used internally by Orbix:

- C++.
 - Java.
-

Remote connections to the Orbix security service

Orbix plug-ins can communicate with the Orbix security service through an IIOP/TLS connection.

Standalone or embedded deployment

The *iSF server module* can be packaged in the following different ways:

- Standalone deployment (default)—the iSF server module is packaged as a standalone server process, the *Orbix security service*, that services requests through a CORBA interface (IIOP or IIOP/TLS).
 - Embedded deployment—the iSF server module is packaged as a JAR library that can be loaded directly into a Java application. In this case, service requests are made as local calls.
-

iSF adapter API

Integration with third-party enterprise security systems is facilitated by the *iSF adapter API* that enables the Orbix security service to delegate security operations to other security systems.

iSF adapters

Orbix provides several ready-made adapters that are implemented with the iSF adapter API. The following adapters are available:

- LDAP adapter.
- File adapter (demonstration only—not supported in production environments).

iSF Server Development Kit

Overview

The iSF server development kit (SDK) enables you to implement custom extensions to the iSF. The iSF SDK is divided into the following parts:

- [iSF adapter SDK](#).
- [iSF client SDK](#).

iSF adapter SDK

The iSF adapter SDK provides an API implementing custom iSF adapters. Using this API, you can integrate any enterprise security system with the iSF.

This API is available in both C++ and Java.

iSF client SDK

The iSF client SDK provides an API for Orbix to access the iSF server module's core functionality directly (usually through remote calls).

This is a private API intended only for internal use by Orbix.

Secure Applications

Overview

This section explains how applications from various technology domains are integrated into the Orbix Security Framework.

In this section

This section contains the following subsections:

ART Security Plug-Ins	page 11
Secure CORBA Applications	page 13

ART Security Plug-Ins

Overview

To participate in the Orbix Security Framework, applications load one or more of the ART security plug-ins. Because Orbix is built using a common ART platform, an identical set of security plug-ins are used across the different technology domains of CORBA and Web services. This has the advantage of ensuring maximum security compatibility between these different technology domains.

What is ART?

Orbix's Adaptive Runtime Technology (ART) is a modular framework for constructing distributed systems, based on a lightweight core and an open-ended set of *plug-ins*. ART is the underlying technology in Orbix.

Security plug-ins

An application can load any of the following security plug-ins to enable particular security features and participate in the Orbix Security Framework:

- [IIOP/TLS](#).
 - [HTTPS](#).
 - [CSiv2](#).
 - [GSP](#).
-

IIOP/TLS

The IIOP/TLS plug-in provides applications with the capability to establish secure connections using IIOP over a TLS transport. Authentication is also performed using X.509 certificates. For example, this plug-in is used by CORBA applications.

HTTPS

The HTTPS plug-in provides the capability to establish secure connections using HTTP over a TLS transport. Authentication is also performed using X.509 certificates. For example, this plug-in is used by the Web container to enable secure communications with Web clients.

CSlv2

The Common Secure Interoperability (CSlv2) plug-in provides support for authentication based on a username and password. The CSlv2 plug-in also enables applications to forward usernames or security tokens to other applications over an IIOP or IIOP/TLS connection.

GSP

The GSP plug-in provides an authorization capability for the iSF—that is, the capability to restrict access to certain methods, operations, or attributes, based on the configuration values stored in an external *action-role mapping* XML file. The GSP plug-in works in tandem with the Orbix security service to realize a complete system of role-based access control.

Secure CORBA Applications

Overview

Figure 2 shows how the security plug-ins in a CORBA application cooperate to provide security for the application.

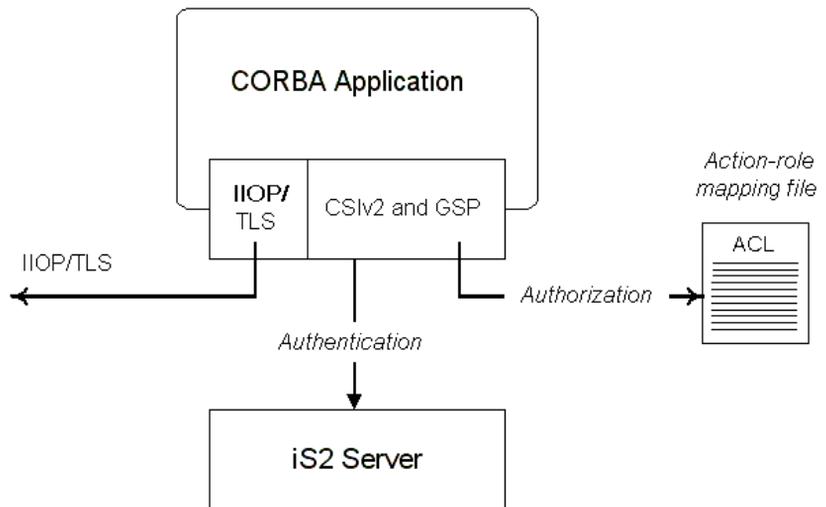


Figure 2: Security Plug-Ins in a CORBA Application

IIOPTLS plug-in in CORBA a application

The IIOPTLS plug-in enables the CORBA application to establish connections secured by SSL/TLS. This layer of security is essential for providing data encryption.

CSlv2 plug-in in a CORBA application

The CSlv2 plug-in provides CORBA applications with the following features:

- The capability to log in with a username and password.
- Screening incoming IIOp invocations by making sure that the username/password combination is correct.
- Transmission of a username/password/domain combination to other applications.
- Transmission of a username or security token to other applications.

GSP plug-in in a CORBA application

The GSP plug-in restricts access to a CORBA server's operations and attributes, only allowing user's with certain specified roles to proceed with an invocation.

Administering the iSF

Overview

This section provides an overview of the main aspects of configuring and administering the iSF.

In this section

This section contains the following subsections:

Overview of iSF Administration	page 16
Secure ASP Services	page 18

Overview of iSF Administration

Overview

There are several different aspects of iSF administration to consider, as follows:

- [Orbix configuration file](#).
- [iSF properties file](#).
- [Enterprise security service administration](#).
- [Access control lists](#).

Orbix configuration file

The Orbix configuration file, *DomainName.cfg* (or, alternatively, the CFR service), is used to configure the security policies for all of the applications and services in a particular location domain. For example, the following kinds of security policy are specified in the Orbix configuration file:

- The list of security plug-ins to be loaded by an application.
- Whether an application accepts both secure and insecure connections, or secure connections only.
- The name of the iSF authorization realm to which an application belongs.

These are just some of the security policies that can be configured—see [“Security Configuration” on page 251](#).

iSF properties file

The iSF properties file is used to configure the core properties of the Orbix security service. This file primarily configures the properties of an iSF adapter that connects to an enterprise security backend. This file also configures the optional single sign-on and authorization manager features.

Enterprise security service administration

Because the Orbix security service is capable of integrating with a third-party enterprise security service, you can continue to use the native third-party administration tools for your chosen enterprise security service. These tools would be used to administer user accounts, including such data as usernames, passwords, user groups, and roles.

Access control lists

To complete a system of role-based access control, it is necessary to provide individual applications with an access control list (ACL) file that is responsible for mapping user roles to particular permissions.

For example, the ACL associated with a CORBA server could specify that only a specified set of roles are allowed to invoke a particular IDL operation.

There is one type of ACL file used within the iSF, as follows:

- Action-role mapping (proprietary format).

Secure ASP Services

Overview

When you create a secure location domain, all of the standard ASP services are secure by default. The default configuration can be used to test sample applications, but is not genuinely secure. Before the ASP services can be used in a real deployment, it is necessary to customize the security configuration.

Customizing the security configuration

For a real deployment, certain aspects of the security configuration for ASP services would be customized, as follows:

- X.509 certificates associated with ASP services—the sample certificates initially associated with the ASP services must all be replaced, because they are not secure.
- Default security policies—for the ASP services might need to be changed before deployment.

Part 2

Orbix Security Framework Administration

In this part

This part contains the following chapters:

Securing Applications and Services	page 31
Managing Access Control Lists	page 61

Transport Layer Security

Transport Layer Security provides encryption and authentication mechanisms for your Orbix system.

In this chapter

This chapter discusses the following topics:

What does Orbix Provide?	page 22
How TLS Provides Security	page 24

What does Orbix Provide?

Security plug-ins

Orbix provides the core security infrastructure to a distributed system based on Orbix's Adaptive Runtime Technology (ART). It is implemented as a symmetric set of plug-ins for Orbix. When the security plug-ins are installed in an application, the communication layers consist of the CORBA standard Internet Inter-ORB Protocol (IIOP), layered above TLS and TCP/IP.

Transport Layer Security

Transport Layer Security (TLS) is an IETF Open Standard. It is based on, and is the successor to, Secure Sockets Layer (SSL), long the standard for secure communications.

The TLS Protocol provides the most critical security features to help you preserve the privacy and integrity of your system:

- Authentication (based on RSA with X.509v3 certificates).
 - Encryption (based on, for example, AES, Triple DES, IDEA).
 - Message integrity (SHA1, SHA256).
 - A framework that allows new cryptographic algorithms to be incorporated into the TLS specification.
-

CORBA Security Level 2

Orbix is based on the CORBA Security Level 2 policies and API's (RTF 1.7). It implements a set of policies from the CORBA specification that enable you to control encryption and authentication at a fine level.

Added-value policies and APIs

Orbix also has added-value policies and APIs that provide more control for SSL/TLS applications than provided by CORBA Security.

Security-unaware and security-aware applications

There are two basic approaches to using security in your applications:

- *Security-unaware applications*—Modify the Orbix configuration to enable and configure security for your application. This approach to security is completely transparent to the application, requiring no code changes or recompilation.

- *Security-aware applications*—In addition to modifying the Orbix configuration to enable security, you can customize application security using both the standard CORBA security API and the Orbix added-value APIs.

How TLS Provides Security

Basic TLS security features

TLS provides the following security for communications across TCP/IP connections:

- Authentication** This allows an application to verify the identity of another application with which it communicates.
- Privacy** This ensures that data transmitted between applications can not be eavesdropped on or understood by a third party.
- Integrity** This allows applications to detect if data was modified during transmission.

In this section

This section contains the following subsections:

Authentication in TLS	page 25
Certificates in TLS Authentication	page 27
Privacy of TLS Communications	page 28
Integrity of TLS Communications	page 29

Authentication in TLS

Public key cryptography

TLS uses Rivest Shamir Adleman (RSA) public key cryptography for authentication. In public key cryptography, each application has an associated public key and private key. Data encrypted with the public key can be decrypted only with the private key. Data encrypted with the private key can be decrypted only with the public key.

Public key cryptography allows an application to prove its identity by encoding data with its private key. As no other application has access to this key, the encoded data must derive from the true application. Any application can check the content of the encoded data by decoding it with the application's public key.

The TLS Handshake Protocol

Consider the example of two applications, a client and a server. The client connects to the server and wishes to send some confidential data. Before sending application data, the client must ensure that it is connected to the required server and not to an impostor.

When the client connects to the server, it confirms the server identity using the TLS handshake protocol. A simplified explanation of how the client executes this handshake in order to authenticate the server is as follows:

Stage	Description
1	The client initiates the TLS handshake by sending the initial TLS handshake message to the server.
2	The server responds by sending its <i>certificate</i> to the client. This certificate verifies the server's identity and contains the certificate's public key.
3	The client extracts the public key from the certificate and encrypts a symmetric encryption algorithm session key with the extracted public key.

Stage	Description
4	The server uses its private key to decrypt the encrypted session key which it will use to encrypt and decrypt application data passing to and from the client. The client will also use the shared session key to encrypt and decrypt messages passing to and from the server.

Optimized handshake

The TLS protocol permits a special optimized handshake in which a previously established session can be resumed. This has the advantage of not needing expensive private key computations. The TLS handshake also facilitates the negotiation of ciphers to be used in a connection.

Client authentication

The TLS protocol also allows the server to authenticate the client. Client authentication, which is supported by Orbix, is optional in TLS communications.

Certificates in TLS Authentication

Purpose of certificates

A public key is transmitted as part of a certificate. The certificate is used to ensure that the submitted public key is, in fact, the public key that belongs to the submitter. The client checks that the certificate has been digitally signed by a certification authority (CA) that the client explicitly trusts.

Certification authority

A CA is a trusted authority that verifies the validity of the combination of entity name and public key in a certificate. You must specify trusted CAs in order to use Orbix.

X.509 certificate format

The International Telecommunications Union (ITU) recommendation, X.509, defines a standard format for certificates. TLS authentication uses X.509 certificates to transfer information about an application's public key.

An X.509 certificate includes the following data:

- The name of the entity identified by the certificate.
- The public key of the entity.
- The name of the certification authority that issued the certificate.

The role of a certificate is to match an entity name to a public key.

Access to certificates

According to the TLS protocol, it is unnecessary for applications to have access to all certificates. Generally, each application only needs to access its own certificate and the corresponding issuing certificates. Clients and servers supply their certificates to applications that they want to contact during the TLS handshake. The nature of the TLS handshake is such that there is nothing insecure in receiving the certificate from an as yet untrusted peer. The certificate will be checked to make sure that it has been digitally signed by a trusted CA and the peer will have to prove its identity during the handshake.

Privacy of TLS Communications

Establishing a symmetric key

Immediately after authentication, the client sends an encoded data value to the server (using the server's public key). This unique session encoded value is a key to a symmetric cryptographic algorithm. Only the server is able to decode this data (using the corresponding private key).

Symmetric cryptography

A symmetric cryptographic algorithm is an algorithm in which a single key is used to encode and decode data. Once the server has received such a key from the client, all subsequent communications between the applications can be encoded using the agreed symmetric cryptographic algorithm. This feature strengthens TLS security.

Examples of symmetric cryptographic algorithms used to maintain privacy in TLS communications are 3DES and AES.

Integrity of TLS Communications

Message authentication code

The authentication and privacy features of TLS ensure that applications can exchange confidential data that cannot be understood by an intermediary. However, these features do not protect against the modification of encrypted messages transmitted between applications.

To detect if an application has received data modified by an intermediary, TLS adds a message authentication code (MAC) to each message. This code is computed by applying a function to the message content and the secret key used in the symmetric cryptographic algorithm.

Guaranteeing message integrity

An intermediary cannot compute the MAC for a message without knowing the secret key used to encrypt it. If the message is corrupted or modified during transmission, the message content will not match the MAC. TLS automatically detects this error and rejects corrupted messages.

Securing Applications and Services

This chapter describes how to enable security in the context of the Orbix Security Framework for different types of applications and services.

In this chapter

This chapter discusses the following topics:

Connecting to an Off-Host iS2 Server	page 32
Securing CORBA Applications	page 33
Securing Orbix Services	page 59
Caching of Credentials	page 60

Connecting to an Off-Host iS2 Server

Overview

Many of the examples in this chapter use the Orbix Security Framework (iSF), which requires access to the iS2 server. Because Orbix Mainframe 6.x does not support the iS2 server on z/OS, it is necessary to run the iS2 server off-host (for example, on UNIX or Windows) and connect your mainframe applications to this off-host service.

Configure and run the iS2 server on another host

For detailed instructions on how to configure and run an iS2 server off-host, see the version of the *Orbix Security Guide* for the UNIX and Windows platforms.

Modify the Orbix configuration on z/OS

To configure your z/OS applications to use an off-host iS2 server, perform the following steps:

1. On the host where the iS2 server is running (UNIX or Windows), open the local Orbix configuration file, *iS2Domain.cfg*, and look for a configuration entry of the following form:

```
# Orbix Configuration File
...
initial_references:IT_SecurityService:reference =
    "IOR:0100...";
```

Copy the `initial_references:IT_SecurityService:reference` entry from the *iS2Domain.cfg* file.

2. On the z/OS host, open the Orbix configuration file located in the `HLQ.DOMAINS PDS` and paste the `initial_references:IT_SecurityService:reference` setting from the iS2 host (either adding the entry or replacing an existing entry).

Securing CORBA Applications

Overview

Using Orbix's modular ART technology, you make a CORBA application secure just by configuring it to load the relevant security plug-ins. This section describes how to load and configure security plug-ins to reach the appropriate level of security for your CORBA applications.

In this section

This section contains the following subsections:

Overview of CORBA Security	page 34
Securing Communications with SSL/TLS	page 36
Specifying Fixed Ports for SSL/TLS Connections	page 46
Securing Two-Tier CORBA Systems with iSF	page 48
Securing Three-Tier CORBA Systems with iSF	page 53

Overview of CORBA Security

Overview

There are two main components of security for CORBA applications: IIOP over SSL/TLS (IIOP/TLS), which provides secure communication between client and server; and the iSF, which is concerned with higher-level security features such as authentication and authorization.

The following combinations are recommended:

- IIOP/TLS only—for a pure SSL/TLS security solution.
- IIOP/TLS and iSF—for a highly scalable security solution, based on username/password client authentication.

CORBA applications and iSF

Figure 3 shows the main features of a secure CORBA application in the context of the iSF.

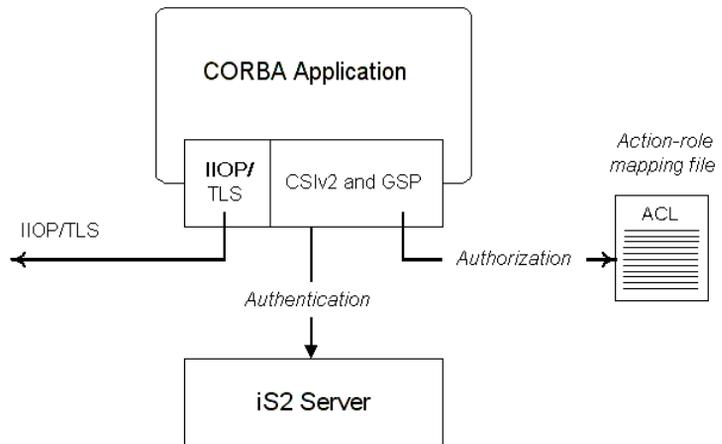


Figure 3: A Secure CORBA Application within the iSF

Security plug-ins

Within the iSF, a CORBA application becomes fully secure by loading the following plug-ins:

- [IIOP/TLS plug-in](#)
 - [CSlv2 plug-in \(Java only\)](#)
 - [GSP plug-in](#)
-

IIOP/TLS plug-in

The IIOP/TLS plug-in, `iiop_tls`, enables a CORBA application to transmit and receive IIOP requests over a secure SSL/TLS connection. This plug-in can be enabled independently of the other two plug-ins.

See [“Securing Communications with SSL/TLS” on page 36](#) for details on how to enable IIOP/TLS in a CORBA application.

CSlv2 plug-in (Java only)

The CSlv2 plug-in, `csi`, provides a mechanism for propagating username/password credentials between CORBA applications. When the CSlv2 plug-in is combined with the GSP plug-in, the username and password are forwarded to a central iS2 server to be authenticated. This plug-in is needed to support the iSF.

Note: The IIOP/TLS plug-in also provides a client authentication mechanism (based on SSL/TLS and X.509 certificates). The SSL/TLS and CSlv2 authentication mechanisms are independent of each other and can be used simultaneously.

GSP plug-in

The GSP plug-in provides an authentication capability for the iSF. When the GSP plug-in is loaded into an Orbix application, CSI credentials are automatically forwarded to the iS2 server to be authenticated. This plug-in is needed to support the iSF.

Securing Communications with SSL/TLS

Overview

This section describes how to configure an application to use SSL/TLS security. In this section, it is assumed that your initial configuration comes from a secure location domain

WARNING: The default certificates used in the CORBA configuration samples are for demonstration purposes only and are completely insecure. You must generate your own custom certificates for use in your own CORBA applications.

Configuration samples

[Appendix D on page 317](#) includes a variety of SSL/TLS configuration scopes that you can use as a starting point for configuring your own applications. The following sample SSL/TLS configuration scopes are available:

- `demos.tls.secure_client_with_no_cert`
- `demos.tls.secure_client_with_cert`
- `demos.tls.semi_secure_client_with_cert`
- `demos.tls.semi_secure_client_with_no_cert`
- `demos.tls.secure_server_no_client_auth`
- `demos.tls.secure_server_request_client_auth`
- `demos.tls.secure_server_enforce_client_auth`
- `demos.tls.semi_secure_server_no_client_auth`
- `demos.tls.semi_secure_server_request_client_auth`
- `demos.tls.semi_secure_server_enforce_client_auth`

Secure client terminology

The terminology used to describe the preceding client configuration scopes is explained in [Table 1](#).

Table 1: *Terminology Describing Secure Client Sample Configurations*

Scope Name Prefix/Suffix	Description
<code>secure_client</code>	The client opens only secure SSL/TLS connections to the server. If the server does not support secure connections, the connection attempt will fail.

Table 1: Terminology Describing Secure Client Sample Configurations

Scope Name Prefix/Suffix	Description
semi_secure_client	<p>The type of connection opened by the client depends on the disposition of the server:</p> <ul style="list-style-type: none"> • If the server is insecure (listening only on an insecure IIOp port), an insecure connection is established. • If the server is secure (listening only on a secure IIOp/TLS port), a secure SSL/TLS connection is established. • If the server is semi-secure (listening on both an IIOp port and on an IIOp/TLS port), the type of connection established depends on the client's <code>binding:client_binding_list</code>. <ul style="list-style-type: none"> ◆ If, in the client's <code>binding:client_binding_list</code>, a binding with the <code>IIOp</code> interceptor appears before a binding with the <code>IIOp_TLS</code> interceptor, an insecure connection is established. ◆ Conversely, if a binding with the <code>IIOp_TLS</code> interceptor appears before a binding with the <code>IIOp</code> interceptor, a secure connection is established.
with_no_cert	No X.509 certificate is associated with the client (at least, not through configuration).
with_cert	An X.509 certificate is associated with the client by setting the principal sponsor configuration variables.

Secure server terminology

The terminology used to describe the preceding server configuration scopes is explained in [Table 2](#).

Table 2: *Terminology Describing Secure Server Sample Configurations*

Scope Name Prefix/Suffix	Description
secure_server	The server accepts only secure SSL/TLS connection attempts. If a remote client does not support secure connections, the connection attempt will fail.
semi_secure_server	The server accepts both secure and insecure connection attempts by remote clients.
no_client_auth	The server does not support client authentication over SSL/TLS. That is, during an SSL/TLS handshake, the server will not request the client to send an X.509 certificate.
request_client_auth	The server allows a connecting client the option of either authenticating itself or not authenticating itself using an X.509 certificate.
enforce_client_auth	The server requires a connecting client to authenticate itself using an X.509 certificate.

Outline of a sample configuration scope

For example, the `demos.tls.secure_server_no_client_auth` configuration defines a server configuration that is secured by SSL/TLS but does not expect clients to authenticate themselves. This configuration has the following outline:

```
# Orbix Configuration File
...
# General configuration at root scope.
...
demos {
  ...
  tls {
    # Common SSL/TLS configuration settings.
    ...
    secure_server_no_client_auth {
      # Specific server configuration settings.
      ...
    };
  };
};
...

```

Three significant groups of configuration variables contribute to the `secure_server_no_client_auth` configuration, as follows:

1. *General configuration at root scope*—these configuration settings are common to *all* applications, whether secure or insecure.
2. *Common SSL/TLS configuration settings*—specify the basic settings for SSL/TLS security. In particular, the `orb_plugins` list defined in this scope includes the `iiop_tls` plug-in.
3. *Specific server configuration settings*—define the settings specific to the `secure_server_no_client_auth` configuration.

Sample client configuration

For example, consider a secure SSL/TLS client whose configuration is modelled on the `demos.tls.secure_client_with_no_cert` configuration.

[Example 1](#) shows how to configure such a sample client.

Example 1: Sample SSL/TLS Client Configuration

```
# Orbix Configuration File
...
# General configuration at root scope.
...
my_secure_apps {
  # Common SSL/TLS configuration settings.
  # (copied from 'demos.tls')
1  orb_plugins = ["local_log_stream", "iiop_profile", "giop",
  "iiop_tls"];
2
  binding:client_binding_list = ["OTS+POA_Coloc", "POA_Coloc",
  "OTS+TLS_Coloc+POA_Coloc", "TLS_Coloc+POA_Coloc",
  "OTS+GIOP+IIOP", "GIOP+IIOP", "OTS+GIOP+IIOP_TLS",
  "GIOP+IIOP_TLS"];
3
  policies:mechanism_policy:protocol_version = "TLS_V1_2";
  policies:mechanism_policy:ciphersuites =
  ["RSA_WITH_RC4_128_SHA",
  "RSA_WITH_AES_256_CBC_SHA256",
  "RSA_WITH_RC4_128_MD5",
  "RSA_WITH_AES_256_CBC_SHA",
  "RSA_WITH_DES_CBC_SHA",
  "RSA_WITH_3DES_EDE_CBC_SHA",
  "RSA_WITH_AES_128_CBC_SHA",
  "RSA_WITH_AES_128_CBC_SHA256"];
4
  event_log:filters = ["IT_ATLI_TLS=*", "IT_IIOP=*",
  "IT_IIOP_TLS=*", "IT_TLS=*"];
  ...
  my_client {
    # Specific SSL/TLS client configuration settings
    # (copied from 'demos.tls.secure_client_with_no_cert')
5    principal_sponsor:use_principal_sponsor = "false";
6
    policies:client_secure_invocation_policy:requires =
    ["Confidentiality", "EstablishTrustInTarget"];
    policies:client_secure_invocation_policy:supports =
    ["Confidentiality", "Integrity", "DetectReplay",
    "DetectMisordering", "EstablishTrustInTarget"];
  }
}
```

Example 1: *Sample SSL/TLS Client Configuration*

```

};
};
...

```

The preceding client configuration can be described as follows:

1. Make sure that the `orb_plugins` variable in this configuration scope includes the `iiop_tls` plug-in.

Note: For fully secure applications, you should *exclude* the `iiop` plug-in (insecure IIOp) from the ORB plug-ins list. This renders the application incapable of making insecure IIOp connections.

For semi-secure applications, however, you should *include* the `iiop` plug-in in the ORB plug-ins list.

If you plan to use the full Orbix Security Framework, you should include the `gsp` plug-in in the ORB plug-ins list as well—see [“Securing Two-Tier CORBA Systems with ISF” on page 48](#).

2. Make sure that the `binding:client_binding_list` variable includes bindings with the `IIOp_TLS` interceptor. You can use the value of the `binding:client_binding_list` shown here.

If you plan to use the full Orbix Security Framework, you should use the `binding:client_binding_list` as shown in [“Client configuration” on page 49](#) instead.

3. The SSL/TLS mechanism policy specifies the default security protocol version and the available cipher suites—see [“Specifying Cipher Suites” on page 119](#).
4. This line enables console logging for security-related events, which is useful for debugging and testing. Because there is a performance penalty associated with this option, you might want to comment out or delete this line in a production system.
5. The SSL/TLS principal sponsor is a mechanism that can be used to specify an application’s own X.509 certificate. Because this client configuration does not use a certificate, the principal sponsor is disabled by setting `principal_sponsor:use_principal_sponsor` to `false`.

6. The following two lines set the *required* options and the *supported* options for the client secure invocation policy. In this example, the policy is set as follows:
 - ◆ Required options—the options shown here ensure that the client can open only secure SSL/TLS connections.
 - ◆ Supported options—the options shown include all of the association options, except for the `EstablishTrustInClient` option. The client cannot support `EstablishTrustInClient`, because it has no X.509 certificate.

Sample server configuration

Generally speaking, it is rarely necessary to configure such a thing as a *pure server* (that is, a server that never makes any requests of its own). Most real servers are applications that act in both a server role and a client role. Hence, the sample server described here is a hybrid of the following two demonstration configurations:

- `demos.tls.secure_server_request_client_auth`
- `demos.tls.secure_client_with_cert`

[Example 2](#) shows how to configure such a sample server.

Example 2: Sample SSL/TLS Server Configuration

```

# Orbix Configuration File
...
# General configuration at root scope.
...
my_secure_apps {
1   # Common SSL/TLS configuration settings.
    # (copied from 'demos.tls')
    ...
    my_server {
        # Specific SSL/TLS server configuration settings
        # (from 'demos.tls.secure_server_request_client_auth')
2       policies:target_secure_invocation_policy:requires =
        ["Confidentiality"];
        policies:target_secure_invocation_policy:supports =
        ["EstablishTrustInClient", "Confidentiality", "Integrity",
        "DetectReplay", "DetectMisordering",
        "EstablishTrustInTarget"];
3
4       principal_sponsor:use_principal_sponsor = "true";
        principal_sponsor:auth_method_id = "security_label";

```

Example 2: Sample SSL/TLS Server Configuration

```

5     principal_sponsor:auth_method_data = ["label=RingLabel"];
6
    # Choose an SAF key ring or an HFS key database:
    # plugins:systemssl_toolkit:saf_keyring = "SAFKeyRing";
    # plugins:systemssl_toolkit:hfs_keyring_filename =
    "HFSKeyRing";

    # Specific SSL/TLS client configuration settings
    # (copied from 'demos.tls.secure_client_with_cert')
7     policies:client_secure_invocation_policy:requires =
["Confidentiality", "EstablishTrustInTarget"];
    policies:client_secure_invocation_policy:supports =
["Confidentiality", "Integrity", "DetectReplay",
"DetectMisordering", "EstablishTrustInClient",
"EstablishTrustInTarget"];
    };
};
...

```

The preceding server configuration can be described as follows:

1. You can use the same common SSL/TLS settings here as described in the preceding [“Sample client configuration” on page 40](#)
2. The following two lines set the *required* options and the *supported* options for the target secure invocation policy. In this example, the policy is set as follows:
 - ◆ Required options—the options shown here ensure that the server accepts only secure SSL/TLS connection attempts.
 - ◆ Supported options—all of the target association options are supported.
3. A server must always be associated with an X.509 certificate. Hence, this line enables the SSL/TLS principal sponsor, which specifies a certificate for the application.
4. This line specifies that the X.509 certificate is contained in an RACF key ring or an HFS database. For more details, see [“Specifying an Application’s Own Certificate” on page 139](#).

5. Replace the X.509 certificate, by editing the `label` option in the `principal_sponsor:auth_method_data` configuration variable to point at a custom X.509 certificate in an RACF key ring or HFS database. For more details, see [“Specifying an Application’s Own Certificate” on page 139](#).
6. Uncomment one of the following lines, setting one of the variables to choose either an RACF key ring or a HFS key database as the source of X.509 certificates. See [“Specifying the Source of Certificates for an z/OS Application” on page 95](#) for more details.
7. The following two lines set the *required* options and the *supported* options for the client secure invocation policy. In this example, the policy is set as follows:
 - ◆ Required options—the options shown here ensure that the application can open only secure SSL/TLS connections to other servers.
 - ◆ Supported options—all of the client association options are supported. In particular, the `EstablishTrustInClient` option is supported when the application is in a client role, because the application has an X.509 certificate.

Mixed security configurations

Most realistic secure server configurations are mixed in the sense that they include both server settings (for the server role), and client settings (for the client role). When combining server and client security settings for an application, you must ensure that the settings are consistent with each other.

For example, consider the case where the server settings are secure and the client settings are insecure. To configure this case, set up the server role as described in [“Sample server configuration” on page 42](#). Then configure the client role by adding (or modifying) the following lines to the `my_secure_apps.my_server` configuration scope:

```
orb_plugins = ["local_log_stream", "iiop_profile", "giop",
              "iiop", "iiop_tls"];
policies:client_secure_invocation_policy:requires =
  ["NoProtection"];
policies:client_secure_invocation_policy:supports =
  ["NoProtection"];
```

The first line sets the ORB plug-ins list to make sure that the `iiop` plug-in (enabling insecure IOP) is included. The `NoProtection` association option, which appears in the required and supported client secure invocation policy, effectively disables security for the client role.

Customizing SSL/TLS security policies

You can, optionally, customize the SSL/TLS security policies in various ways. For details, see the following references:

- [“Configuring SSL/TLS Secure Associations” on page 103.](#)
- [“Configuring SSL/TLS Authentication” on page 131.](#)

Specifying Fixed Ports for SSL/TLS Connections

Overview

Orbix allows you to specify a fixed IP port on which a server listens for SSL/TLS connections. This subsection provides an overview of the programming and configuration requirements for setting IIOPTLS fixed ports.

POA policies required for setting fixed ports

The main prerequisite for configuring fixed ports is that a CORBA developer programs the application to create a POA instance with the following policies:

- `PortableServer::LifespanPolicy`—the value of this POA policy should be set to `PERSISTENT`, indicating that the objects managed by this POA can outlive the server process.
 - `IT_CORBA::WellKnownAddressingPolicy`—the value of this POA policy is a string that defines a well-known addressing prefix, `<wka_prefix>`, for host/port configuration variables that an administrator can edit in the Orbix configuration.
 - `IT_PortableServer::PersistenceModePolicy`—the value of this POA policy can be set to either of the following values:
 - ◆ `DIRECT_PERSISTENCE`, indicating that the POA is configured to receive connection attempts *directly* from clients. The server listens on the fixed port (well-known address) and exports IORs containing its own host and fixed port.
 - ◆ `INDIRECT_PERSISTENCE`, indicating that connection attempts will be redirected to the server by the locator service. The server listens on the fixed port (well-known address), but exports IORs containing the locator's host and port.
-

Programming the required POA policies

For details of how to program POA policies, see the *CORBA Programmer's Guide*.

Fixed port configuration variables

The following IIOp/TLS configuration variables can be set for a POA that supports the well-known addressing policy with the `<wka_prefix>` prefix:

```
<wka_prefix>:iiop_tls:host = "<host>";
```

Specifies the hostname, `<host>`, to publish in the IIOp/TLS profile of server-generated IORs.

```
<wka_prefix>:iiop_tls:port = "<port>";
```

Specifies the fixed IP port, `<port>`, on which the server listens for incoming IIOp/TLS messages. This port value is also published in the IIOp/TLS profile of generated IORs.

```
<wka_prefix>:iiop_tls:listen_addr = "<host>";
```

Restricts the IIOp/TLS listening point to listen only on the specified host, `<host>`. It is generally used on multi-homed hosts to limit incoming connections to a particular network interface.

```
<wka_prefix>:iiop_tls:addr_list =
["<optional_plus_sign><host>:<port>", ... ];
```

In the context of server clustering, this configuration variable specifies a list of host and port combinations, `<host>:<port>`, for the `<wka_prefix>` persistent POA instance.

One of the host and port combinations, `<host>:<port>` (lacking a + prefix), specifies the POA's own listening point. The other host and port combinations, `+<host>:<port>` (including a + prefix), specify the listening points for other servers in the cluster.

Note: The `*:addr_list` variable takes precedence over the other host/port configuration variables (`*:host`, `*:port`, and `*:listen_addr`).

Securing Two-Tier CORBA Systems with iSF

Overview

This section describes how to secure a two-tier CORBA system using the iSF. The client supplies username/password authentication data which is then authenticated on the server side. The following configurations are described in detail:

- [Client configuration.](#)
- [Target configuration.](#)

Prerequisites

Before implementing this scenario on the z/OS platform, you must configure your domain to use an off-host iS2 server.

See [“Connecting to an Off-Host iS2 Server” on page 32.](#)

Two-tier CORBA system

[Figure 4](#) shows a basic two-tier CORBA system in the iSF, featuring a client and a target server.

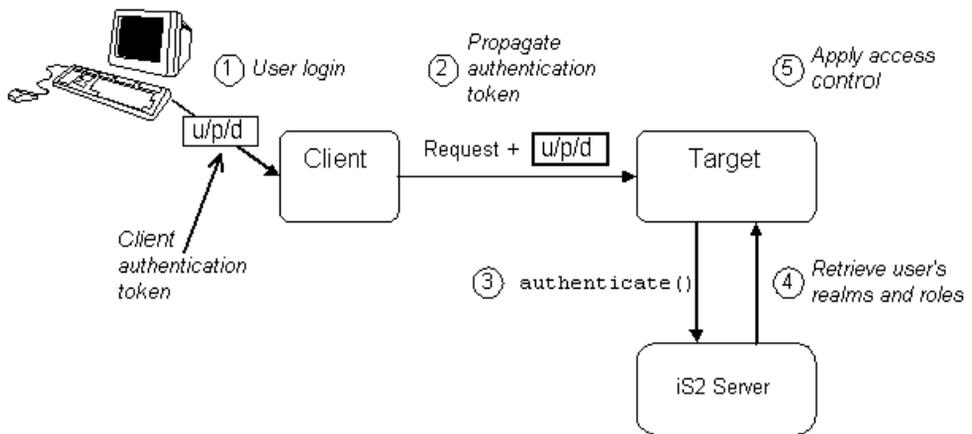


Figure 4: Two-Tier CORBA System in the iSF

Scenario description

The scenario shown in [Figure 4 on page 48](#) can be described as follows:

Stage	Description
1	The user enters a username, password, and domain name on the client side (user login). Note: The domain name is currently ignored by the iSF.
2	When the client makes a remote invocation on the server, the iSF transmits the username/password/domain authentication data to the target along with the invocation request.
3	The server authenticates the received username and password by calling out to the external iS2 server.
4	If authentication is successful, the iS2 server returns a successful status.

Client configuration

The CORBA client from [Figure 4 on page 48](#) can be configured as shown in [Example 3](#).

Example 3: *Configuration of a CORBA client in the iSF*

```

# Orbix Configuration File
...
# General configuration at root scope.
...
my_secure_apps {
1   # Common SSL/TLS configuration settings.
    ...
2   # Common iSF configuration settings.
    orb_plugins = ["local_log_stream", "iiop_profile", "giop",
3   "iiop_tls", "ots", "gsp"];
    binding:client_binding_list = ["GIOP+EGMIOP",
4   "OTS+TLS_Coloc+POA_Coloc", "TLS_Coloc+POA_Coloc",
    "OTS+POA_Coloc", "POA_Coloc", "GIOP+SHMIOP",
    "CSI+OTS+GIOP+IIOP_TLS", "OTS+GIOP+IIOP_TLS",
    "CSI+GIOP+IIOP_TLS", "GIOP+IIOP_TLS", "CSI+OTS+GIOP+IIOP",
    "OTS+GIOP+IIOP", "CSI+GIOP+IIOP", "GIOP+IIOP"];
    binding:server_binding_list = ["CSI+GSP+OTS", "CSI+GSP",
    "CSI+OTS", "CSI"];
    ...
    my_client {

```

Example 3: Configuration of a CORBA client in the iSF

```

5      # Specific SSL/TLS configuration settings.
      ...
      # Specific iSF configuration settings.
6      policies:csi:auth_over_transport:client_supports =
["EstablishTrustInClient"];

7      principal_sponsor:csi:use_principal_sponsor = "true";
      principal_sponsor:csi:auth_method_id = "GSSUPMech";
      principal_sponsor:csi:auth_method_data = [];
    };
  };
  ...

```

The preceding client configuration can be explained as follows:

1. The SSL/TLS configuration variables common to all of your applications can be placed here—see [“Securing Communications with SSL/TLS” on page 36](#) for details of the SSL/TLS configuration.
2. Make sure that the `orb_plugins` variable in this configuration scope includes both the `iiop_tls` and the `gsp` plug-ins in the order shown.
3. Make sure that the `binding:client_binding_list` variable includes bindings with the CSI interceptor. You can use the value of the `binding:client_binding_list` shown here.
4. Make sure that the `binding:server_binding_list` variable includes bindings with both the CSI and GSP interceptors. You can use the value of the `binding:server_binding_list` shown here.
5. The SSL/TLS configuration variables specific to the CORBA client can be placed here—see [“Securing Communications with SSL/TLS” on page 36](#).
6. This configuration setting specifies that the client supports sending username/password authentication data to a server.

7. The next three lines specify that the client uses the CSI principal sponsor to obtain the user's authentication data. With the configuration as shown, the user would be prompted to enter the username and password when the client application starts up.

Note: If the client runs on the z/OS platform, you would have to specify the CSI username and password explicitly in the configuration file. z/OS cannot prompt the user for a username and a password.

For more details on the CSI principal sponsor, see [“Providing a Username and Password” on page 168](#).

Target configuration

The CORBA target server from [Figure 4 on page 48](#) can be configured as shown in [Example 4](#).

Example 4: Configuration of a Second-Tier Target Server in the iSF

```
# Orbix Configuration File
...
# General configuration at root scope.
...
my_secure_apps {
    # Common SSL/TLS configuration settings.
    ...
    # Common iSF configuration settings.
    orb_plugins = [ ..., "iiop_tls", "gsp", ... ];
    binding:client_binding_list = [ ... ];
    binding:server_binding_list = [ ... ];
    ...
    my_two_tier_target {
1      # Specific SSL/TLS configuration settings.
        ...
2      # Specific iSF configuration settings.
        policies:csi:auth_over_transport:target_supports =
3      ["EstablishTrustInClient"];
        policies:csi:auth_over_transport:target_requires =
4      ["EstablishTrustInClient"];
        policies:csi:auth_over_transport:server_domain_name =
        "DEFAULT";
```

Example 4: Configuration of a Second-Tier Target Server in the iSF

```

5      # iSF client configuration settings.
      policies:csi:auth_over_transport:client_supports =
        ["EstablishTrustInClient"];

      principal_sponsor:csi:use_principal_sponsor = "true";
      principal_sponsor:csi:auth_method_id = "GSSUPMech";
      principal_sponsor:csi:auth_method_data =
        ["username=Username", "password=Pass", domain="DEFAULT"];
      };
};

```

The preceding target server configuration can be explained as follows:

1. The SSL/TLS configuration variables specific to the CORBA target server can be placed here—see [“Securing Communications with SSL/TLS” on page 36](#).
2. This configuration setting specifies that the target server *supports* receiving username/password authentication data from the client.
3. This configuration setting specifies that the target server *requires* the client to send username/password authentication data.
4. The `server_domain_name` configuration variable sets the server’s CSIv2 authentication domain name. This setting is ignored by the iSF.
5. You should also set iSF client configuration variables in the server configuration scope, because a secure server application usually behaves as a secure client of the core CORBA services. For example, almost all CORBA servers need to contact both the locator service and the CORBA naming service.

Note: The value of the `principal_sponsor:csi:auth_method_data` configuration variable must be set explicitly in the configuration file on the z/OS platform.

Securing Three-Tier CORBA Systems with iSF

Overview

This section describes how to secure a three-tier CORBA system using the iSF. In this scenario there is a client, an intermediate server, and a target server. The intermediate server is configured to propagate the client identity when it invokes on the target server in the third tier. The following configurations are described in detail:

- [Intermediate configuration.](#)
- [Target configuration.](#)

Prerequisites

Before implementing this scenario on the z/OS platform, you must configure your domain to use an off-host iS2 server.

See [“Connecting to an Off-Host iS2 Server” on page 32.](#)

Three-tier CORBA system

[Figure 5](#) shows a basic three-tier CORBA system in the iSF, featuring a client, an intermediate server and a target server.

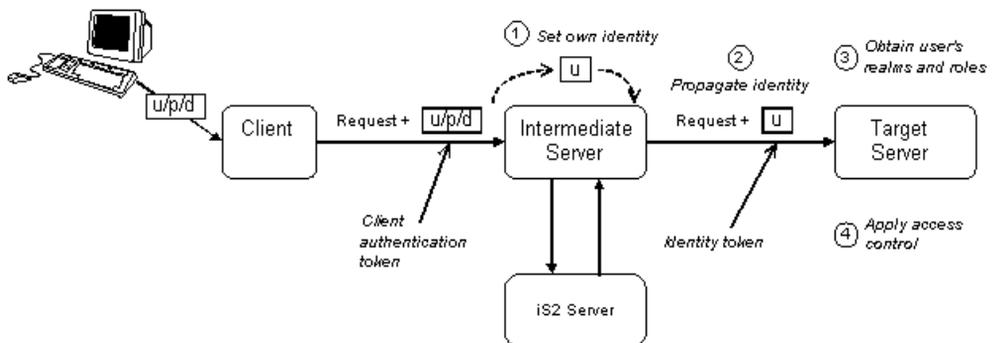


Figure 5: *Three-Tier CORBA System in the iSF*

Scenario description

The second stage of the scenario shown in [Figure 5 on page 53](#) (intermediate server invokes an operation on the target server) can be described as follows:

Stage	Description
1	The intermediate server sets its own identity by extracting the user identity from the received username/password credentials. Hence, the intermediate server assumes the same identity as the client.
2	When the intermediate server makes a remote invocation on the target server, the iSF also transmits the user identity data to the target.
3	The target server then obtains the user's realms and roles.
4	The iSF controls access to the target's IDL interfaces by consulting an <i>action-role mapping file</i> to determine what the user is allowed to do.

Client configuration

The client configuration for the three-tier scenario is identical to that of the two-tier scenario, as shown in ["Client configuration" on page 49](#).

Intermediate configuration

The CORBA intermediate server from [Figure 5 on page 53](#) can be configured as shown in [Example 5](#).

Example 5: Configuration of a Second-Tier Intermediate Server in the iSF

```
# Orbix Configuration File
...
# General configuration at root scope.
...
my_secure_apps {
  # Common SSL/TLS configuration settings.
  ...
  # Common iSF configuration settings.
  orb_plugins = [ ..., "iiop_tls", "gsp", ... ];
  binding:client_binding_list = [ ... ];
  binding:server_binding_list = [ ... ];
  ...
}
```

Example 5: Configuration of a Second-Tier Intermediate Server in the iSF

```

1      my_three_tier_intermediate {
        # Specific SSL/TLS configuration settings.
        ...
        # Specific iSF configuration settings.
2      policies:csi:attribute_service:client_supports =
["IdentityAssertion"];

3      policies:csi:auth_over_transport:target_supports =
["EstablishTrustInClient"];
4      policies:csi:auth_over_transport:target_requires =
["EstablishTrustInClient"];
5      policies:csi:auth_over_transport:server_domain_name =
"DEFAULT";

6      # iSF client configuration settings.
        policies:csi:auth_over_transport:client_supports =
["EstablishTrustInClient"];

        principal_sponsor:csi:use_principal_sponsor = "true";
        principal_sponsor:csi:auth_method_id = "GSSUPMech";
        principal_sponsor:csi:auth_method_data =
["username=Username", "password=Pass", domain="DEFAULT"];
    };
};

```

The preceding intermediate server configuration can be explained as follows:

1. The SSL/TLS configuration variables specific to the CORBA intermediate server can be placed here—see [“Securing Communications with SSL/TLS” on page 36](#).
2. This configuration setting specifies that the intermediate server is capable of propagating the identity it receives from a client. In other words, the server is able to assume the identity of the client when invoking operations on third-tier servers.
3. This configuration setting specifies that the intermediate server *supports* receiving username/password authentication data from the client.
4. This configuration setting specifies that the intermediate server *requires* the client to send username/password authentication data.
5. The `server_domain_name` configuration variable sets the server’s CSIV2 authentication domain name. This setting is ignored by the iSF.

6. You should also set iSF client configuration variables in the intermediate server configuration scope, because a secure server application usually behaves as a secure client of the core CORBA services. For example, almost all CORBA servers need to contact both the locator service and the CORBA naming service.

Note: The value of the `principal_sponsor:csi:auth_method_data` configuration variable must be set explicitly in the configuration file on the z/OS platform.

Target configuration

The CORBA target server from [Figure 5 on page 53](#) can be configured as shown in [Example 6](#).

Example 6: Configuration of a Third-Tier Target Server in the iSF

```
# Orbix Configuration File
...
# General configuration at root scope.
...
my_secure_apps {
    # Common SSL/TLS configuration settings.
    ...
    # Common iSF configuration settings.
    orb_plugins = [ ..., "iiop_tls", "gsp", ... ];
    binding:client_binding_list = [ ... ];
    binding:server_binding_list = [ ... ];
    ...
    my_three_tier_target {
        # Specific SSL/TLS configuration settings.
        ...
1      policies:iiop_tls:target_secure_invocation_policy:requires
2      = ["Confidentiality", "DetectMisordering", "DetectReplay",
        "Integrity", "EstablishTrustInClient"];
3      policies:iiop_tls:certificate_constraints_policy =
        [ConstraintString1, ConstraintString2, ...];

        # Specific iSF configuration settings.
4      policies:csi:attribute_service:target_supports =
        ["IdentityAssertion"];
```

Example 6: Configuration of a Third-Tier Target Server in the iSF

```

5      # iSF client configuration settings.
      policies:csi:auth_over_transport:client_supports =
["EstablishTrustInClient"];

      principal_sponsor:csi:use_principal_sponsor = "true";
      principal_sponsor:csi:auth_method_id = "GSSUPMech";
      principal_sponsor:csi:auth_method_data =
["username=Username", "password=Pass", domain="DEFAULT"];
    };
};

```

The preceding target server configuration can be explained as follows:

1. The SSL/TLS configuration variables specific to the CORBA target server can be placed here—see [“Securing Communications with SSL/TLS” on page 36](#).
2. It is recommended that the target server require its clients to authenticate themselves using an X.509 certificate. For example, the intermediate server (acting as a client of the target) would then be required to send an X.509 certificate to the target during the SSL/TLS handshake.

You can specify this option by including the `EstablishTrustInClient` association option in the target secure invocation policy, as shown here (thereby overriding the policy value set in the outer configuration scope).

3. In addition to the preceding step, it is also advisable to restrict access to the target server by setting a certificate constraints policy, which allows access only to those clients whose X.509 certificates match one of the specified constraints—see [“Applying Constraints to Certificates” on page 144](#).

Note: The motivation for limiting access to the target server is that clients of the target server obtain a special type of privilege: propagated identities are granted access to the target server without the target server performing authentication on the propagated identities. Hence, the target server trusts the intermediate server to do the authentication on its behalf.

4. This configuration setting specifies that the target server supports receiving propagated user identities from the client.
5. You should also set iSF client configuration variables in the target server configuration scope, because a secure server application usually behaves as a secure client of the core CORBA services. For example, almost all CORBA servers need to contact both the locator service and the CORBA naming service.

Note: The value of the `principal_sponsor:csi:auth_method_data` configuration variable must be set explicitly in the configuration file on the z/OS platform.

Securing Orbix Services

Overview

In a secure system, all Orbix services should be capable of servicing secure connections. A minimal system typically includes the following secure services:

- Locator,
- Node daemon,
- Naming service,
- Interface repository (IFR),
- IMS/CICS adapters.

Additionally, your system might also require certificates for the events, notification, and OTS services.

Configuring the Orbix services

Before deploying the Orbix services, you must customize the security configuration, replacing demonstration certificates by custom certificates and so on. The procedure for securing Orbix services is similar to the procedure for securing regular CORBA applications.

See [“Securing CORBA Applications” on page 33](#).

Caching of Credentials

Overview

To improve the performance of servers within the Orbix Security Framework, the GSP plug-in implements caching of credentials (that is, the authentication and authorization data received from the iS2 server).

The GSP credentials cache reduces a server's response time by reducing the number of remote calls to the iS2 security service. On the first call from a given user, the server calls iS2 and caches the received credentials. On subsequent calls from the same user, the cached credentials are used, thereby avoiding a remote call to iS2.

Cache time-out

The cache can be configured to time-out credentials, forcing the server to call iS2 again after using cached credentials for a certain period.

Cache size

The cache can also be configured to limit the number of stored credentials.

Configuration variables

The following variables configure the credentials cache in the context of the Orbix Security Framework:

`plugins:gsp:authentication_cache_size`

The maximum number of credentials stored in the authentication cache. If this size is exceeded the oldest credential in the cache is removed.

A value of -1 (the default) means unlimited size. A value of 0 means disable the cache.

`plugins:gsp:authentication_cache_timeout`

The time (in seconds) after which a credential is considered *stale*. Stale credentials are removed from the cache and the server must re-authenticate with iS2 on the next call from that user.

A value of -1 (the default) means an infinite time-out. A value of 0 means disable the cache.

Managing Access Control Lists

The Orbix Security Framework defines access control lists (ACLs) for mapping roles to resources. The ACLs are specific to particular technology domains, such as CORBA. They can be deployed either together with each secure server or centrally in the Orbix security service.

In this chapter

This chapter discusses the following topics:

CORBA ACLs	page 62
Centralized ACL	page 68

CORBA ACLs

Overview

This section discusses the ACL files that control access to IDL operations and attributes in a CORBA server. The ACL files for CORBA servers provide role-based access control with granularity down to the level of IDL operations, and attributes.

In this section

This section contains the following subsections:

Overview of CORBA ACL Files	page 63
CORBA Action-Role Mapping ACL	page 64

Overview of CORBA ACL Files

Action-role mapping file

The action-role mapping file is an XML file that specifies which user roles have permission to perform specific actions on the server (that is, invoking specific IDL operations and attributes).

GSP plug-in

The GSP plug-in is a component of the ISF that provides support for action-role mapping. This plug-in must be loaded in order to use the action-role mapping ACL file (see [Appendix A](#) for details of how to configure the GSP plug-in).

File restrictions

The following restrictions apply to the use of action role mapping files on z/OS:

- File system—XML ACL files must be stored in the hierarchical file system (USS).
- XML encoding—the XML must use an ASCII-based encoding. The XML data itself must be ASCII-based (for example, UTF-8), as well as the encoding attribute in the XML prolog (if present). This restriction is imposed by the underlying IBM for XML Toolkit (Xerces C++ parser), which is unable to handle the inclusion of the DTD file when an EBCDIC encoding is used.

CORBA Action-Role Mapping ACL

Overview

This subsection explains how to configure the action-role mapping ACL file for CORBA applications. Using an action-role mapping file, you can specify that access to IDL operations and attributes is restricted to specific roles.

File location

In your Orbix configuration file, the `plugins:gsp:action_role_mapping_file` configuration variable specifies the location URL of the action-role mapping file, `action_role_mapping.xml`, for a CORBA server. For example:

```
# Orbix Configuration File
...
my_server_scope {
    plugins:gsp:action_role_mapping_file =
        "file:///security_admin/action_role_mapping.xml";
};
```

Example IDL

For example, consider how to set the operation and attribute permissions for the IDL interface shown in [Example 7](#).

Example 7: *Sample IDL for CORBA ACL Example*

```
// IDL
module Simple
{
    interface SimpleObject
    {
        void call_me();
        attribute string foo;
    };
};
```

Example action-role mapping

[Example 8](#) shows how you might configure an action-role mapping file for the `Simple::SimpleObject` interface given in the preceding [Example 7](#) on [page 64](#).

Example 8: CORBA Action-Role Mapping Example

```

1 <?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE secure-system SYSTEM
    "InstallDir/etc/domains/Domain/actionrolemapping.dtd">
2 <secure-system>
  <allow-unlisted-interfaces>false</allow-unlisted-interfaces>
3
4 <action-role-mapping>
5   <server-name>gsp_basic_test.server</server-name>
6   <interface>
7     <name>IDL:Simple/SimpleObject:1.0</name>
      <action-role>
8       <action-name>call_me</action-name>
        <role-name>corba-developer</role-name>
        <role-name>guest</role-name>
      </action-role>
      <action-role>
        <action-name>_get_foo</action-name>
        <role-name>corba-developer</role-name>
        <role-name>guest</role-name>
      </action-role>
    </interface>
  </action-role-mapping>
</secure-system>

```

The preceding action-role mapping example can be explained as follows:

1. If the directory containing the `actionrolemapping.dtd` file includes spaces, the spaces should be replaced by `%20` in the `<!DOCTYPE>` tag.
2. The `<allow-unlisted-interfaces>` tag specifies the default access that applies to interfaces not explicitly listed in the action-role mapping file. The tag contents can have the following values:
 - ◆ `true`—for any interfaces not listed, access is allowed for all roles. If the remote user is unauthenticated (in the sense that no GSSUP credentials are sent by the client), access is also allowed.

- ◆ `false`—for any interfaces not listed, access is denied for all roles. Unauthenticated users are also denied access. This is the default.
3. The `<action-role-mapping>` tag contains all of the permissions that apply to a particular server application.
 4. The `<server-name>` tag specifies the ORB name that is used by the server in question. The value of this tag must match the ORB name exactly.

Note: The ORB name also determines which configuration scopes are read by the server. See the *Administrator's Guide* for details.

5. The `<interface>` tag contains all of the access permissions for one particular IDL interface.
6. The `<name>` tag identifies the IDL interface using the interface's OMG repository ID. The repository ID normally consists of the characters `IDL:` followed by the fully scoped name of the interface (using `/` instead of `::` as the scoping character), followed by the characters `:1.0`. Hence, the `Simple::SimpleObject` IDL interface is identified by the `IDL:Simple/SimpleObject:1.0` repository ID.

Note: The form of the repository ID can also be affected by various `#pragma` directives appearing in the IDL file. A commonly used directive is `#pragma prefix`.

For example, the `CosNaming::NamingContext` interface in the naming service module, which uses the `omg.org` prefix, has the following repository ID: `IDL:omg.org/CosNaming/NamingContext:1.0`

7. The `call_me` action name corresponds to the `call_me()` operation in the `Simple::SimpleObject` interface. The action name corresponds to the GIOP on-the-wire form of the operation name (usually the same as it appears in IDL).

8. The `_get_foo` action name corresponds to the `foo` attribute accessor. In general, any read/write attribute, *AttributeName*, has the following action names:

- ◆ `_get_AttributeName` (for the attribute accessor)
- ◆ `_set_AttributeName` (for the attribute modifier)

In general, the accessor or modifier action names correspond to the GIOP on-the-wire form of the attribute accessor or modifier.

Action-role mapping DTD

The syntax of the action-role mapping file is defined by the action-role mapping DTD. See [“Action-Role Mapping DTD” on page 327](#) for details.

Centralized ACL

Overview

By default, a secure Orbix application is configured to store its ACL file locally. Hence, in a large deployment, ACL files might be scattered over many hosts, which could prove to be a nuisance for administrators.

An alternative approach, as described in this section, is to configure your secure applications to use a centralized ACL repository. This allows you to administer all of the ACL data in one place, making it easier to update and maintain.

In this section

This section contains the following subsections:

Local ACL Scenario	page 69
Centralized ACL Scenario	page 71
Customizing Access Control Locally	page 77

Local ACL Scenario

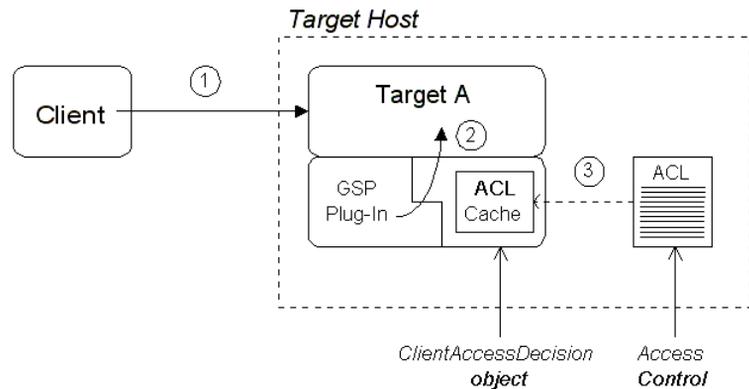
Overview

This section briefly describes the behavior of a secure server whose operations are protected by a local ACL file.

Local ACL scenario

Figure 6 shows an outline of the local ACL scenario, where the ACL file is stored on the same host as the target server. You configure the server to load the ACL file from the local file system by setting the `plugins:gsp:action_role_mapping_file` variable in the target server's configuration scope.

Figure 6: *Local ACL Scenario*



Scenario description

The local ACL scenario shown in [Figure 6](#) can be described as follows:

Stage	Description
1	The client invokes an operation on the secure target server, requiring an access decision to be made on the server side.
2	The GSP plug-in calls a function on the internal <code>ClientAccessDecision</code> object to check whether the current user has permission to invoke the current operation.
3	If this is the first access decision required by the target server, the <code>ClientAccessDecision</code> object reads the contents of the local ACL file (as specified by the <code>plugins:gsp:action_role_mapping_file</code> variable) and stores the ACL data in a cache. For all subsequent access decisions, the <code>ClientAccessDecision</code> object reads the cached ACL data for efficiency.

Centralized ACL Scenario

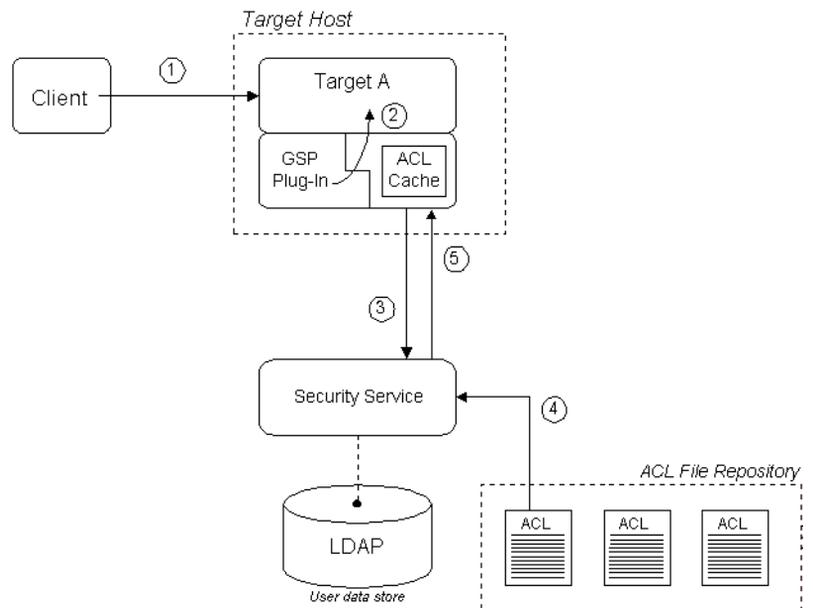
Overview

From an administrative point of view, it is often more convenient to gather ACL files onto a central host, rather than leaving them scattered on different hosts. The *centralized ACL* feature enables you to create such a central repository of ACL files. The ACL files are stored on the same host as the Orbix security service, which serves up ACL data to remote Orbix servers on request.

Centralized ACL scenario

Figure 7 shows an outline of a centralized ACL scenario, where the ACL files are stored on the same host as the Orbix security service.

Figure 7: *Centralized ACL scenario*



Scenario description

The centralized ACL scenario shown in [Figure 7](#) can be described as follows:

Stage	Description
1	The client invokes an operation on the secure target server, requiring an access decision to be made on the server side.
2	The GSP plug-in calls a function on the internal <code>ClientAccessDecision</code> object to check whether the current user has permission to invoke the current operation.
3	If this is the first access decision required by the target server, the <code>ClientAccessDecision</code> object contacts the Orbix security service to obtain the ACL data. For all subsequent access decisions, the <code>ClientAccessDecision</code> object reads the cached ACL data for efficiency.
4	When the security service is requested to provide ACL data, it selects the appropriate ACL file from its repository of ACL files. By default, the Orbix security service selects the ACL file whose ORB name (as specified in the <code><server-name></code> tag) matches that of the request.
5	The security service returns the ACL data in the form of an XML string, which is then cached by the <code>ClientAccessDecision</code> object.

Modify the Orbix configuration file

To configure an application (such as the target server shown in [Figure 7 on page 71](#)) to use a centralized ACL, you must modify its configuration scope as shown in [Example 9](#). In this example, it is assumed that the application's ORB name is `my_secure_apps.my_two_tier_target`.

Example 9: *Configuration of a Second-Tier Target Server in the iSF*

```
# Orbix Configuration File
...
# General configuration at root scope.
...
my_secure_apps {
```

Example 9: Configuration of a Second-Tier Target Server in the iSF

```

...
my_two_tier_target {
  ...
  plugins:gsp:authorization_realm = "AuthzRealm";
1  # plugins:gsp:action_role_mapping_file = "ActionRoleURL";
2  plugins:gsp:authorization_policy_store_type =
3  "centralized";
  plugins:gsp:authorization_policy_enforcement_point =
  "local";
  };
};

```

The preceding Orbix configuration can be described as follows:

1. The `plugins:gsp:action_role_mapping_file` setting is ignored when you have centralized ACL enabled. You can either comment out this line, as shown here, or delete it.
2. Setting the `plugins:gsp:authorization_policy_store_type` variable to `centralized` configures the application to retrieve its ACL data from the Orbix security service (which is then stored in a local cache).
3. Setting the `plugins:gsp:authorization_policy_enforcement_point` variable to `local` specifies that the ACL logic is implemented locally (in the target server). Currently, this is the only option that is supported.

Modify the `is2.properties` file

To configure the Orbix security service to support centralized ACL, you should edit its `is2.properties` (normally located in the `OrbixInstallDir/etc/domains/DomainName` directory) to add or modify the following settings:

```

# is2.properties File for the Orbix Security Service
...
com.iona.isp.authz.adapters=file
com.iona.isp.authz.adapter.file.class=com.iona.security.is2AzAdapter.multifile.MultiFileAzAdapter
com.iona.isp.authz.adapter.file.param.filelist=ACLFileListFile;

```

The `ACLFileListFile` is the name of a file (specified in the local file format) which contains a list of the centrally stored ACL files.

Create an ACL file list file

The ACL file list file is a list of filenames, each line of which has the following format:

```
[ACLKey=]ACLFileName
```

A file name can optionally be preceded by an ACL key and an equals sign, *ACLKey=*, if you want to select the file by ACL key (see “[Selection by ACL key](#)” on page 76). The ACL file, *ACLFileName*, is specified using an absolute pathname in the local file format.

Note: On Windows, you should replace backslashes by forward slashes in the pathname.

For example, on Windows you could specify a list of ACL files as follows:

```
U:/orbix_security/etc/acl_files/server_A.xml
U:/orbix_security/etc/acl_files/server_B.xml
U:/orbix_security/etc/acl_files/server_C.xml
```

Selecting the ACL file

When the Orbix security service responds to a request to provide ACL data, it chooses an ACL file using one of the following selection criteria:

- [Selection by ORB name.](#)
- [Selection by override value.](#)
- [Selection by ACL key.](#)

Selection by ORB name

The default selection criterion is *selection by ORB name*. The target application includes its ORB name in the request it sends to the security service. The security service then selects the data from the ACL file which includes a `<server-name>` tag with the specified ORB name.

Note: The security service reads and returns *all* of the data from the selected ACL file. Even if the ACL file contains multiple `<server-name>` tags labelled by different ORB names, the data from the enclosing `<action-role-mapping>` tags with non-matching ORB names are also returned.

For example, if the application's ORB name is `my_secure_apps.my_two_tier_target`, the security service will select the data from the ACL file containing the following `<server-name>` tag:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE secure-system SYSTEM "DTDFileForOrbixACL">
<secure-system>
  <action-role-mapping>
    <server-name>my_secure_apps.my_two_tier_target</server-name>
    ...
  </action-role-mapping>
  ...
</secure-system>
```

Selection by override value

Alternatively, you can use *selection by override value* to override the value of the ORB name sent to the Orbix security service. The override value must be set in the Orbix configuration using the `plugins:gsp:acl_policy_data_id` variable.

For example, suppose you want to select ACL data that has the ORB name, `my_secure_apps.my_two_tier_target.alt_acl`. You would specify the override value using the `plugins:gsp:acl_policy_data_id` variable as follows:

```
# Orbix Configuration File
...
# Add this line to the application's configuration scope
plugins:gsp:acl_policy_data_id =
  "my_secure_apps.my_two_tier_target.alt_acl";
```

The security service would then select the data from the ACL file containing the following `<server-name>` tag:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE secure-system SYSTEM "DTDFileForOrbixACL">
<secure-system>
  <action-role-mapping>

    <server-name>my_secure_apps.my_two_tier_target.alt_acl</server-name>
    ...
  </action-role-mapping>
  ...
</secure-system>
```

Selection by ACL key

A more flexible system of selection is *selection by ACL key*. In this case, the application specifies an ACL key in its Orbix configuration and the security service matches this key to an entry in the ACL file list file.

For example, consider an application that defines an ACL key, `bank_data`, in its configuration scope. You would specify the key using the `plugins:gsp:acl_policy_data_id` variable as follows:

```
# Orbix Configuration File
...
# Add this line to the application's configuration scope
plugins:gsp:acl_policy_data_id = "aclkey:bank_data";
```

The security service then selects the entry from the ACL file list labelled with the `bank_data` key:

```
U:/orbix_security/etc/acl_files/server_A.xml
U:/orbix_security/etc/acl_files/server_B.xml
bank_data=U:/orbix_security/etc/acl_files/server_C.xml
```

Customizing Access Control Locally

Overview

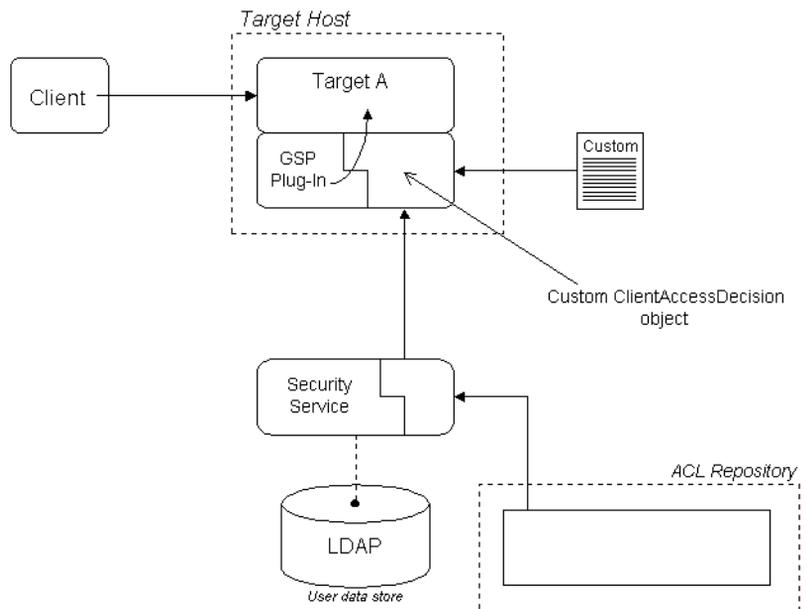
Orbix allows you to customize access control locally by implementing a plug-in that overrides the implementation of the `ClientAccessDecision` object. This gives you complete control over the access decision logic in an Orbix application.

Note: Detailed instructions on how to implement a `ClientAccessDecision` plug-in are not provided here. Because this task requires a detailed understanding of Orbix plug-ins, we recommend that you contact Micro Focus for further assistance.

Custom `ClientAccessDecision` in an Orbix application

Figure 8 shows an outline of an ACL scenario, where the default `ClientAccessDecision` object is replaced by a customized implementation.

Figure 8: *Custom `ClientAccessDecision` in an Orbix Application*



Scenario variants

Replacing the `ClientAccessDecision` object with a customized implementation effectively gives you complete control over the access decision logic in an Orbix application. The system shown in [Figure 8](#) can be adapted to a variety of scenarios, as follows:

- Storing the ACL data locally, but using a customized file format.
- Customizing both the `ClientAccessDecision` object and the `ServerAccessDecision` object to implement a centralized ACL with custom features. In particular, this approach would enable you to store and transmit ACL data in a custom format.
- Retrieving ACL data from a custom server. In this case, you could have a centralized ACL repository that bypasses the Orbix security service.

Managing Certificates

TLS authentication uses X.509 certificates—a common, secure and reliable method of authenticating your application objects. This chapter explains how you can create X.509 certificates that identify your Orbix applications.

In this chapter

This chapter contains the following sections:

What are X.509 Certificates?	page 80
Certification Authorities	page 82
Certificate Chaining	page 85
PKCS#12 Files	page 87
Managing Certificates on z/OS	page 88

What are X.509 Certificates?

Role of certificates

An X.509 certificate binds a name to a public key value. The role of the certificate is to associate a public key with the identity contained in the X.509 certificate.

Integrity of the public key

Authentication of a secure application depends on the integrity of the public key value in the application's certificate. If an impostor replaced the public key with its own public key, it could impersonate the true application and gain access to secure data.

To prevent this form of attack, all certificates must be signed by a *certification authority (CA)*. A CA is a trusted node that confirms the integrity of the public key value in a certificate.

Digital signatures

A CA signs a certificate by adding its *digital signature* to the certificate. A digital signature is a message encoded with the CA's private key. The CA's public key is made available to applications by distributing a certificate for the CA. Applications verify that certificates are validly signed by decoding the CA's digital signature with the CA's public key.

WARNING: Most of the demonstration certificates supplied with Orbix are signed by the CA `abigbank_ca.pem`. This CA is completely insecure because anyone can access its private key. To secure your system, you must create new certificates signed by a trusted CA. This chapter describes the set of certificates required by an Orbix application and shows you how to replace the default certificates.

The contents of an X.509 certificate

An X.509 certificate contains information about the certificate subject and the certificate issuer (the CA that issued the certificate). A certificate is encoded in Abstract Syntax Notation One (ASN.1), a standard syntax for describing messages that can be sent or received on a network.

The role of a certificate is to associate an identity with a public key value. In more detail, a certificate includes:

- X.509 version information.

- A *serial number* that uniquely identifies the certificate.
- A *subject DN* that identifies the certificate owner.
- The *public key* associated with the subject.
- An *issuer DN* that identifies the CA that issued the certificate.
- The digital signature of the issuer.
- Information about the algorithm used to sign the certificate.
- Some optional X.509 v.3 extensions. For example, an extension exists that distinguishes between CA certificates and end-entity certificates.

Distinguished names

A distinguished name (DN) is a general purpose X.500 identifier that is often used in the context of security.

See [“ASN.1 and Distinguished Names” on page 307](#) for more details about DNs.

Certification Authorities

Choice of CAs

A CA must be trusted to keep its private key secure. When setting up an Orbix system, it is important to choose a suitable CA, make the CA certificate available to all applications, and then use the CA to sign certificates for your applications.

There are two types of CA you can use:

- A *commercial CA* is a company that signs certificates for many systems.
- A *private CA* is a trusted node that you set up and use to sign certificates for your system only.

In this section

This section contains the following subsections:

Commercial Certification Authorities	page 83
Private Certification Authorities	page 84

Commercial Certification Authorities

Signing certificates

There are several commercial CAs available. The mechanism for signing a certificate using a commercial CA depends on which CA you choose.

Advantages of commercial CAs

An advantage of commercial CAs is that they are often trusted by a large number of people. If your applications are designed to be available to systems external to your organization, use a commercial CA to sign your certificates. If your applications are for use within an internal network, a private CA might be appropriate.

Criteria for choosing a CA

Before choosing a CA, you should consider the following criteria:

- What are the certificate-signing policies of the commercial CAs?
- Are your applications designed to be available on an internal network only?
- What are the potential costs of setting up a private CA?

Private Certification Authorities

Choosing a CA software package

If you wish to take responsibility for signing certificates for your system, set up a private CA. To set up a private CA, you require access to a software package that provides utilities for creating and signing certificates. Several packages of this type are available.

Choosing a host for a private certification authority

Choosing a host is an important step in setting up a private CA. The level of security associated with the CA host determines the level of trust associated with certificates signed by the CA.

If you are setting up a CA for use in the development and testing of Orbix applications, use any host that the application developers can access. However, when you create the CA certificate and private key, do not make the CA private key available on hosts where security-critical applications run.

Security precautions

If you are setting up a CA to sign certificates for applications that you are going to deploy, make the CA host as secure as possible. For example, take the following precautions to secure your CA:

- Do not connect the CA to a network.
- Restrict all access to the CA to a limited set of trusted users.
- Protect the CA from radio-frequency surveillance using an RF-shield.

Certificate Chaining

Certificate chain

A *certificate chain* is a sequence of certificates, where each certificate in the chain is signed by the subsequent certificate.

Self-signed certificate

The last certificate in the chain is normally a *self-signed certificate*—a certificate that signs itself.

Example

Figure 9 shows an example of a simple certificate chain.

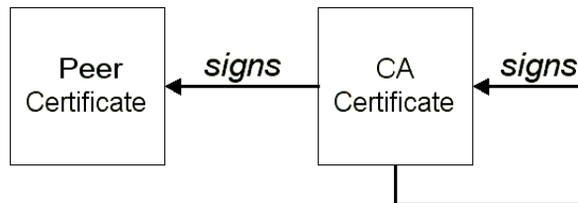


Figure 9: A Certificate Chain of Depth 2

Chain of trust

The purpose of certificate chain is to establish a chain of trust from a peer certificate to a trusted CA certificate. The CA vouches for the identity in the peer certificate by signing it. If the CA is one that you trust (indicated by the presence of a copy of the CA certificate in your root certificate directory), this implies you can trust the signed peer certificate as well.

Certificates signed by multiple CAs

A CA certificate can be signed by another CA. For example, an application certificate may be signed by the CA for the finance department of Micro Focus, which in turn is signed by a self-signed commercial CA. [Figure 10](#) shows what this certificate chain looks like.

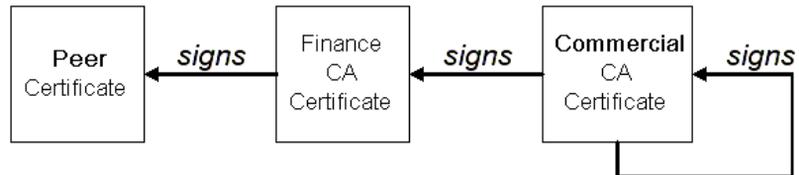


Figure 10: A Certificate Chain of Depth 3

Trusted CAs

An application can accept a signed certificate if the CA certificate for any CA in the signing chain is available in the certificate file in the local root certificate directory.

Maximum chain length policy

You can limit the length of certificate chains accepted by your applications, with the maximum chain length policy. You can set a value for the maximum length of a certificate chain with the `policies:iop_tls:max_chain_length_policy` and `policies:https:max_chain_length_policy` configuration variables for IOP/TLS and HTTPS respectively.

PKCS#12 Files

Contents of a PKCS#12 file

A PKCS#12 file contains the following:

- An X.509 peer certificate (first in a chain).
- All the CA certificates in the certificate chain.
- A private key.

The file is encrypted with a password.

PKCS#12 is an industry-standard format and is used by browsers such as Netscape and Internet Explorer. They are also used in Orbix. Orbix does not support .pem format certificate chains, however.

z/OS Platform

The trusted CA list policy is not used on the z/OS platform if you configure your applications to use SAF key rings.

Managing Certificates on z/OS

Certificate management using RACF

On z/OS, certificates are managed and stored in a different way from other platforms. This section describes the management of certificates on z/OS using RACF. Users of other z/OS security products should refer to the relevant product documentation.

X.509 certificates provide a common, secure and reliable method of authenticating your application objects. If a component of your application must prove its identity during SSL authentication, that component requires a certificate signed by your chosen CA. In a secure system, this always includes the locator, the node daemons, the Orbix utilities, the Orbix services, and your server programs. If you use client authentication, your clients also require certificates.

HFS key databases

It is also possible to use HFS key databases for some of the items discussed below. Key databases are discussed in the IBM manual, *Cryptographic Services System Secure Sockets Layer Programming*. Using a key database is an option in a test environment. However, key databases are currently limited in the types of PKCS#12 certificates they import, so they are not so easy to use with externally provided certificates.

In this section

This section contains the following subsections:

Importing Certificates from Another Platform into RACF	page 89
Creating Certificates for an Application Using RACF	page 94
Specifying the Source of Certificates for an z/OS Application	page 95

Importing Certificates from Another Platform into RACF

Certificate import options

You can obtain certificates using one of the following options:

- Import certificates from another platform.
- Import certificates from a party, such as a public CA.
- Generate certificates using RACF.

This section explains how to import certificates from another platform.

The `RACDCERT` command

This section provides some examples of the `RACDCERT` command usage. A full description of this command can be found in the IBM manual, *z/OS Security Server (RACF) Command Language Reference*. Refer to the manual for details on setting up the permissions in RACF to use the `RACDCERT` commands.

Importing certificates into RACF

To import certificates in to RACF from another platform, perform the following steps:

1

Allocate the datasets on z/OS.

To set up the secure certificates on z/OS in RACF, you need temporary datasets that will contain the certificates transmitted from the other platform. You usually need to create at least two datasets. One is for a text format (PEM) Certification Authority (CA) certificate. The other one is for a binary format (PKCS#12) application certificate. Both datasets need to be variable length record datasets.

The datasets do not have to be very big. The following allocation parameters should be sufficient in most cases:

Organization	PS
Record format	VB
Record length	1024
Block size	32760
Allocated blocks	2
Allocated extents	1

For example, to import some of the demonstration certificates supplied with Orbix on other platforms, you could create the following two datasets:

```
USERID.CERT.IONACA.PEM
USERID.CERT.BANKSRV.P12
```

The following sections use these two names, where *USERID* is your user ID or any suitable top-level name. The first name, *USERID.CERT.IONACA.PEM*, stores the demonstration CA certificate. The second name, *USERID.CERT.BANKSRV.P12*, stores the *bankserver.p12* certificate. However, any suitable dataset names can be used.

2

FTP the certificates into the z/OS datasets.

Below is an example where the two certificates are copied from a UNIX machine to z/OS. An important thing to note is that the PEM (ASCII) format CA certificate is copied in `ascii` mode and that the binary PKCS#12 certificate is copied in `binary` mode. In this example *userid* is the user name and the *hostname* is the z/OS hostname.

```
13:02:34 userid - 15> pwd
.../etc/tls/x509/certs/demos
13:02:34 userid - 15> ftp hostname
Connected to hostname.iona.com.
220-FTPD1 IBM FTP CS V2R8 at hostname.iona.com, 09:26:01 on
    2001-08-15.
220 Connection will close if idle for more than 5 minutes.
Name (hostname:userid):
331 Send password please.
Password:
230 USERID is logged on. Working directory is "USERID".
ftp> ascii
200 Representation type is Ascii NonPrint
ftp> put ca_list1.pem 'USERID.CERT.IONACA.PEM'
200 Port request OK.
125 Storing data set USERID.CERT.IONACA.PEM
250 Transfer completed successfully.
local: ca_list1.pem remote: 'USERID.CERT.IONACA.PEM'
1670 bytes sent in 0.021 seconds (76.46 Kbytes/s)
ftp> bin
200 Representation type is Image
ftp> put bank_server.p12 'USERID.CERT.BANKSRV.P12'
200 Port request OK.
125 Storing data set USERID.CERT.BANKSRV.P12
250 Transfer completed successfully.
local: bank_server.p12 remote: 'USERID.CERT.BANKSRV.P12'
3538 bytes sent in 0.014 seconds (253.10 Kbytes/s)
```

```
ftp> quit
221 Quit command received. Goodbye.
13:02:34 userid - 15>
```

After the FTP transfer, you can inspect the datasets using an editor like ISPF. The CA dataset must be in readable format and looks something like:

```
-----BEGIN CERTIFICATE-----
MIIBjDCCATagAwIBAgIIv5hpmk5TOF8wDQYJKoZIhvcNAQEEBQAwSzELMAkGA1UE
...
...
oudXbfbj1QZQ+TPKvJHe9w==
-----END CERTIFICATE-----
```

The bank server certificate is in binary format and is not readable.

The certificates are now ready to be added to an RACF key ring.

3

Import the certificates into RACF using RACDCERT commands.

The next step is to import the certificates into RACF. The `RACDCERT` command is used for this. The first certificate to import is the CA certificate. The following JCL imports the certificate into RACF:

```
//RACFCERT JOB    (),
//          CLASS=A,
//          MSGCLASS=X,
//          MSGLEVEL=(1,1),
//          NOTIFY=&SYSUID,
//          REGION=0M,
//          TIME=1440
//STEP1 EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
RACDCERT CERTAUTH ADD('USERID.CERT.IONACA.PEM') -
WITHLABEL('ionaca')
/*
```

For the CA certificate, you have to specify `CERTAUTH` so that RACF is aware that the certificate is a CA certificate. Also, case is important, so if `ionaca` is specified in lowercase in this job, the same has to be done in all the other jobs using this label.

The command to import the bank server certificate is:

```
//STEP1 EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
RACDCERT ID(USERID) ADD('USERID.CERT.BANKSRV.P12') -
WITHLABEL('bank_server') PASSWORD('bankserverpass')
/*
```

For PKCS#12 files, a password needs to be supplied. The password is the one used to encrypt the private key in the PKCS#12 file. The certificate private key is then stored in the RACF database and the password does not have to be used again.

It is now possible to view the content of the certificate. Use the following command to verify the content of the certificate:

```
//STEP1 EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
  RACDCERT LIST(LABEL('bank_server'))
/*
```

This displays all kinds of information about the certificate, including the status, the name on the certificate and the dates for which it is valid.

4

Add the certificates to the user key ring.

The final step is to create the user key ring and to add the certificates to the key ring. The first item is to create the key ring. For example, a key ring called `TESTRING` can be created as follows:

```
//STEP1 EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
  RACDCERT ADDRING(TESTRING)
/*
```

The certificates can then be added to the key ring. You have to add both the CA certificate and the user certificate to the key ring. The following command adds the CA certificate:

```
//STEP1 EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
  RACDCERT CONNECT(CERTAUTH LABEL('ionaca') RING(TESTRING))
/*
```

The following command adds the user certificate:

```
//STEP1 EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
  RACDCERT CONNECT(ID(USERID) LABEL('bank_server')
  RING(TESTRING))
/*
```

You can check if both certificates were successfully added by listing the contents of the key ring.

```
//STEP1 EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
  RACDCERT LISTRING (TESTRING)
/*
```

The output should look something like this:

```
RACDCERT LISTRING (TESTRING)
```

Digital ring information for user *USERID*:

Ring:

```
>TESTRING<
Certificate Label Name      Cert Owner      USAGE          DEFAULT
-----
bank_server                 ID (USERID)    PERSONAL       NO
ionaca                      CERTAUTH       CERTAUTH       NO
```

The key ring is now ready for use. You can repeat the preceding steps to add more certificates to RACF and to the key ring, if you wish.

Creating Certificates for an Application Using RACF

Using RACF as a CA

It is also possible to use RACF as a Certification Authority for in-house certificates. There are three steps required to do this:

1. Set up a CA.
2. Use the CA to create signed certificates.
3. Deploy the signed certificates into the user key rings.

References

These steps are fully described in the following IBM manuals:

- *Cryptographic Services System Secure Sockets Layer Programming*
- *Security Server (RACF) - Command Language Reference*

Specifying the Source of Certificates for an z/OS Application

Alternative certificate sources

A source of certificates *must* be specified for every secure z/OS application (both clients and servers). The following alternatives are supported:

- SAF key ring.
- HFS key database.

SAF/RACF key ring

To use an SAF/RACF key ring, `TESTRING`, set the `saf_keyring` configuration variable as follows:

```
plugins:systemssl_toolkit:saf_keyring = "TESTRING";
```

For details of how to create the `TESTRING` key ring, see [“Importing Certificates from Another Platform into RACF” on page 89](#).

Note: When using an SAF key ring, do *not* specify a password or password stash file.

HFS key database

Alternatively, to use a HFS key database, set the `hfs_keyring_filename` configuration variable to specify the key database file. For example, you can specify a `/keyring/key.kdb` database file, as follows:

```
plugins:systemssl_toolkit:hfs_keyring_filename =
  "/keyring/key.kdb";
```

For a description of how to set up a HFS key database, please consult the IBM document *Cryptographic Services System Secure Sockets Layer Programming* from the *Cryptographic Services* bookshelf.

Password for HFS key database

A password must also be specified for the HFS key database. There are two alternatives:

- To specify the password directly in the configuration file, set the `hfs_keyring_file_password` configuration variable, as follows:


```
plugins:systemssl_toolkit:hfs_keyring_file_password =
  "password";
```
- To use a password stash file, `passfile.stash`, set the `hfs_keyring_file_stashfile` configuration variable, as follows:


```
plugins:systemssl_toolkit:hfs_keyring_file_stashfile =
  "passfile.stash";
```

The `passfile.stash` file contains an encrypted password. See the IBM document *Cryptographic Services System Secure Sockets Layer Programming* for details of how to create a password stash file.

Part 3

SSL/TLS Administration

In this part

This part contains the following chapters:

Choosing an SSL/TLS Toolkit	page 99
Managing Certificates	page 79
Configuring SSL/TLS Secure Associations	page 103
Configuring SSL/TLS Authentication	page 131

Choosing an SSL/TLS Toolkit

This chapter describes the SSL/TLS toolkit replaceability feature, which enables you to replace the underlying third-party toolkit that implements the SSL/TLS protocol for Orbix applications.

In this chapter

This chapter contains the following sections:

Toolkit Replaceability	page 100
System SSL Toolkit on z/OS	page 101

Toolkit Replaceability

Overview

In Orbix, the underlying SSL/TLS security layer is provided by a third-party security toolkit. The Orbix security configuration variables and programming APIs wrap the third-party toolkit in order to integrate it with CORBA technology.

Orbix provides a *toolkit replaceability* feature by exploiting Orbix's Adaptive Runtime Technology (ART) to encapsulate third-party SSL/TLS toolkits in an ART plug-in. Using this modular approach, you can replace the SSL/TLS security layer underlying Orbix by specifying a different ART plug-in to load at runtime.

Toolkits for C++ applications

The following SSL/TLS toolkits are currently available for use with Orbix C++ applications:

- [“System SSL Toolkit on z/OS” on page 101.](#)

Custom toolkit plug-in for C++

Orbix also provides an option to develop a custom toolkit plug-in for C++ applications, using the Orbix plug-in development kit (PDK). You can use this feature to integrate any third-party SSL/TLS toolkit with Orbix.

Please contact Professional Services for more details:

<https://www.microfocus.com/en-us/support/consulting-professional-services>

System SSL Toolkit on z/OS

Overview

This section describes how to configure Orbix to use the System SSL toolkit, which is native to the z/OS platform.

Choosing the System SSL toolkit for C++ applications

To ensure that Orbix uses the System SSL toolkit for C++ applications, you must add the settings shown in [Example 10](#) to your Orbix configuration.

Example 10: Configuring Orbix to use the System SSL Toolkit in C++

```
# Orbix configuration file
initial_references:IT_TLS_Toolkit:plugin = "systemssl_toolkit";
plugins:systemssl_toolkit:shlib_name = "ORXSSSL";
```

These settings are specified in the Orbix internal configuration file, `orbix_internal.cfg` file, (the `ORXINTRL` dataset member).

Configuring SSL/TLS Secure Associations

You can govern the behavior of client-server connections by setting configuration variables to choose association options and to specify cipher suites.

In this chapter

This chapter discusses the following topics:

Overview of Secure Associations	page 104
Setting Association Options	page 106
Specifying Cipher Suites	page 119

Overview of Secure Associations

Secure association

Secure association is the CORBA term for any link between a client and a server that enables invocations to be transmitted securely. In practice, a secure association is often realized as a TCP/IP network connection augmented by a particular security protocol (such as TLS) but many other realizations are possible.

In the context of Orbix, secure associations always use TLS.

TLS session

A *TLS session* is the TLS implementation of a secure client-server association. The TLS session is accompanied by a *session state* that stores the security characteristics of the association.

A TLS session underlies each secure association in Orbix.

Colocation

For *colocated invocations*, that is where the calling code and called code share the same address space, Orbix supports the establishment of colocated secure associations. A special interceptor, `TLS_CoLoc`, is provided by the security plug-in to optimize the transmission of secure, colocated invocations.

Configuration overview

The security characteristics of an association can be configured through the following CORBA policy types:

- *Client secure invocation policy*—enables you to specify the security requirements on the client side by setting association options. See [“Choosing Client Behavior” on page 110](#) for details.
- *Target secure invocation policy*—enables you to specify the security requirements on the server side by setting association options. See [“Choosing Target Behavior” on page 112](#) for details.
- *Mechanism policy*—enables you to specify the security mechanism used by secure associations. In the case of TLS, you are required to specify a list of cipher suites for your application. See [“Specifying Cipher Suites” on page 119](#) for details.

Figure 11 illustrates all of the elements that configure a secure association. The security characteristics of the client and the server can be configured independently of each other.

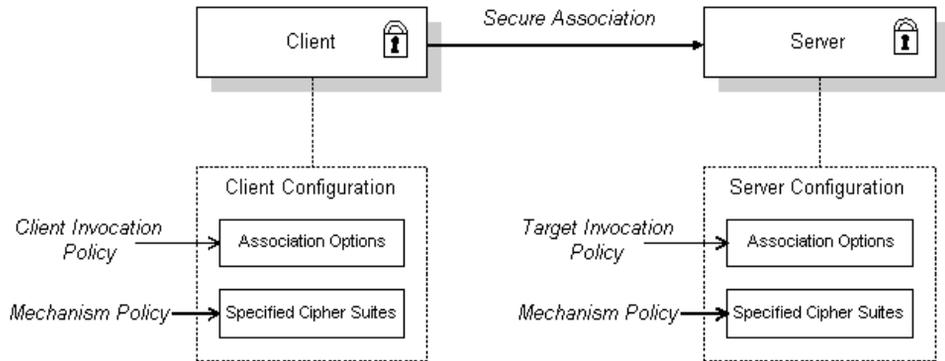


Figure 11: Configuration of a Secure Association

Setting Association Options

Overview

This section explains the meaning of the various SSL/TLS association options and describes how you can use the SSL/TLS association options to set client and server secure invocation policies for both SSL/TLS and HTTPS connections.

In this section

The following subsections discuss the meaning of the settings and flags:

Secure Invocation Policies	page 107
Association Options	page 108
Choosing Client Behavior	page 110
Choosing Target Behavior	page 112
Hints for Setting Association Options	page 114

Secure Invocation Policies

Secure invocation policies

You can set the minimum security requirements of objects in your system with two types of security policy:

- *Client secure invocation policy*—specifies the client association options.
- *Target secure invocation policy*—specifies the association options on a target object.

These policies can only be set through configuration; they cannot be specified programmatically by security-aware applications.

OMG-defined policy types

The client and target secure invocation policies correspond to the following policy types, as defined in the OMG security specification:

- `Security::SecClientSecureInvocation`
- `Security::SecTargetSecureInvocation`

These policy types are, however, not directly accessible to programmers.

Configuration example

For example, to specify that client authentication is required for IIOP/TLS connections, you can set the following target secure invocation policy for your server:

```
# Orbix Configuration File
secure_server_enforce_client_auth
{
    policies:iiop_tls:target_secure_invocation_policy:requires =
["EstablishTrustInClient", "Confidentiality"];

    policies:iiop_tls:target_secure_invocation_policy:supports =
["EstablishTrustInClient", "Confidentiality", "Integrity",
"DetectReplay", "DetectMisordering",
"EstablishTrustInTarget"];

    // Other settings (not shown)...
};
```

Association Options

Available options

You can use *association options* to configure Orbix. They can be set for clients or servers where appropriate. These are the available options:

- `NoProtection`
 - `Integrity`
 - `Confidentiality`
 - `DetectReplay`
 - `DetectMisordering`
 - `EstablishTrustInTarget`
 - `EstablishTrustInClient`
-

NoProtection

Use the `NoProtection` flag to set minimal protection. This means that insecure bindings are supported, and (if the application supports something other than `NoProtection`) the object can accept secure and insecure invocations. This is the equivalent to `SEMI_SECURE` servers in OrbixSSL.

Integrity

Use the `Integrity` flag to indicate that the object supports integrity-protected invocations. Setting this flag implies that your TLS cipher suites support message digests (such as SHA256, SHA1).

Confidentiality

Use the `Confidentiality` flag if your object requires or supports at least confidentiality-protected invocations. The object can support this feature if the cipher suites specified by the `MechanismPolicy` support confidentiality-protected invocations.

DetectReplay

Use the `DetectReplay` flag to indicate that your object supports or requires replay detection on invocation messages. This is determined by characteristics of the supported TLS cipher suites.

DetectMisordering

Use the `DetectMisordering` flag to indicate that your object supports or requires error detection on fragments of invocation messages. This is determined by characteristics of the supported TLS cipher suites.

EstablishTrustInTarget

The `EstablishTrustInTarget` flag is set for client policies only. Use the flag to indicate that your client supports or requires that the target authenticate its identity to the client. This is determined by characteristics of the supported TLS cipher suites. This is normally set for both client `supports` and `requires` unless anonymous cipher suites are supported.

Note: On z/OS, the `EstablishTrustInTarget` policy setting is ignored. The peer client always performs server authentication. This is because the underlying IBM System SSL toolkit always enforces authentication of the target and cannot be configured to do otherwise.

EstablishTrustInClient

Use the `EstablishTrustInClient` flag to indicate that your target object requires the client to authenticate its privileges to the target. This option cannot be required as a client policy.

If this option is supported on a client's policy, it means that the client is prepared to authenticate its privileges to the target. On a target policy, the target supports having the client authenticate its privileges to the target.

Note: Examples of all the common cases for configuring association options can be found in the default Orbix configuration file—see the `demos.tls` scope of the `ASPIInstallDir/etc/domains/DomainName.cfg` configuration file.

Choosing Client Behavior

Client secure invocation policy

The `Security::SecClientSecureInvocation` policy type determines how a client handles security issues.

IIOPTLS configuration

You can set this policy for IIOPTLS connections through the following configuration variables:

```
policies:iioptls:client_secure_invocation_policy:requires
    Specifies the minimum security features that the client requires to
    establish an IIOPTLS connection.
```

```
policies:iioptls:client_secure_invocation_policy:supports
    Specifies the security features that the client is able to support on
    IIOPTLS connections.
```

HTTPS configuration

You can set this policy for HTTPS connections through the following configuration variables:

```
policies:https:client_secure_invocation_policy:requires
    Specifies the minimum security features that the client requires to
    establish a HTTPS connection.
```

```
policies:https:client_secure_invocation_policy:supports
    Specifies the security features that the client is able to support on
    HTTPS connections.
```

Association options

In both cases, you provide the details of the security levels in the form of `AssociationOption` flags—see [“Association Options” on page 108](#) and [Appendix C on page 313](#).

Default value

The default value for the client secure invocation policy is:

```
supports      Integrity, Confidentiality, DetectReplay,
              DetectMisordering, EstablishTrustInTarget
requires      Integrity, Confidentiality, DetectReplay,
              DetectMisordering, EstablishTrustInTarget
```

Example

In the default configuration file, the `demos.tls.bank_client` scope specifies the following association options:

```
# Orbix Configuration File
# In 'demos.tls' scope
  bank_client {
    ...
    policies:iiop_tls:client_secure_invocation_policy:requires =
      ["Confidentiality", "EstablishTrustInTarget"];

    policies:iiop_tls:client_secure_invocation_policy:supports =
      ["Confidentiality", "Integrity", "DetectReplay",
       "DetectMisordering", "EstablishTrustInTarget"];
  };
  ...
};
```

Choosing Target Behavior

Target secure invocation policy

The `Security::SecTargetSecureInvocation` policy type operates in a similar way to the `Security::SecClientSecureInvocation` policy type. It determines how a target handles security issues.

IIOPTLS configuration

You can set the target secure invocation policy for IIOPTLS connections through the following configuration variables:

```
policies:iioptls:target_secure_invocation_policy:requires
```

Specifies the minimum security features that your targets require, before they accept an IIOPTLS connection.

```
policies:iioptls:target_secure_invocation_policy:supports
```

Specifies the security features that your targets are able to support on IIOPTLS connections.

HTTPS configuration

You can set the target secure invocation policy for HTTPS connections through the following configuration variables:

```
policies:https:target_secure_invocation_policy:requires
```

Specifies the minimum security features that your targets require, before they accept a HTTPS connection.

```
policies:https:target_secure_invocation_policy:supports
```

Specifies the security features that your targets are able to support on HTTPS connections.

Association options

In both cases, you can provide the details of the security levels in the form of `AssociationOption` flags—see [“Association Options” on page 108](#) and [Appendix C on page 313](#).

Default value

The default value for the target secure invocation policy is:

```
supports      Integrity, Confidentiality, DetectReplay,
              DetectMisordering, EstablishTrustInTarget
```

```
requires      Integrity, Confidentiality, DetectReplay,
              DetectMisordering
```

Example

In the default configuration file, the `demos.tls.bank_server` scope specifies the following association options:

```
# Orbix Configuration File
# In 'demos.tls' scope
...
bank_server {
  ...
  policies:iiop_tls:target_secure_invocation_policy:requires =
    ["Confidentiality"];

  policies:iiop_tls:target_secure_invocation_policy:supports =
    ["Confidentiality", "Integrity", "DetectReplay",
     "DetectMisordering", "EstablishTrustInTarget"];
  ...
};
...
```

Hints for Setting Association Options

Overview

This section gives an overview of how association options can be used in real applications.

Use the sample scopes

The quickest way to configure a secure SSL/TLS application is by basing the configuration on one of the sample `demo.s.tls` scopes in the `DomainName.cfg` configuration file. In `demo.s.tls`, there are sample scopes that match all of the common use cases for SSL/TLS configuration.

For more details, see [“Configuration samples” on page 36](#).

Rules of thumb

The following rules of thumb should be kept in mind:

- If an association option is *required* by a particular invocation policy, it must also be *supported* by that invocation policy. It makes no sense to require an association option without supporting it.
- It is important to be aware that the secure invocation policies and the security mechanism policy mutually interact with each other. That is, the association options effective for a particular secure association depend on the available cipher suites (see [“Constraints Imposed on Cipher Suites” on page 124](#)).
- The `NoProtection` option must appear alone in a list of *required* options. It does not make sense to require other security options in addition to `NoProtection`.

Types of association option

Association options can be categorized into the following different types, as shown in [Table 3](#).

Table 3: *Description of Different Types of Association Option*

Description	Relevant Association Options
Request or require TLS peer authentication.	EstablishTrustInTarget and EstablishTrustInClient .
Quality of protection.	Confidentiality , Integrity , DetectReplay , and DetectMisordering .
Allow or require insecure connections.	NoProtection .

EstablishTrustInTarget and EstablishTrustInClient

These association options are used as follows:

- [EstablishTrustInTarget](#)—determines whether a server sends its own X.509 certificate to a client during the SSL/TLS handshake. In practice, secure Orbix applications must enable [EstablishTrustInTarget](#), because all of the cipher suites supported by Orbix require it.

The [EstablishTrustInTarget](#) association option should appear in all of the configuration variables shown in the relevant row of [Table 4](#).
- [EstablishTrustInClient](#)—determines whether a client sends its own X.509 certificate to a server during the SSL/TLS handshake. The [EstablishTrustInClient](#) feature is optional and various combinations of settings are possible involving this association option.

The `EstablishTrustInClient` association option can appear in any of the configuration variables shown in the relevant row of [Table 4](#).

Table 4: *Setting `EstablishTrustInTarget` and `EstablishTrustInClient` Association Options*

Association Option	Client side—can appear in...	Server side—can appear in...
<code>EstablishTrustInTarget</code>	<code>policies:client_secure_invocation_policy:supports</code> <code>policies:client_secure_invocation_policy:requires</code>	<code>policies:target_secure_invocation_policy:supports</code>
<code>EstablishTrustInClient</code>	<code>policies:client_secure_invocation_policy:supports</code>	<code>policies:target_secure_invocation_policy:supports</code> <code>policies:target_secure_invocation_policy:requires</code>

Note: The SSL/TLS client authentication step can also be affected by the `policies:allow_unauthenticated_clients_policy` configuration variable. See [“policies” on page 272](#).

Confidentiality, Integrity, DetectReplay, and DetectMisordering

These association options can be considered together, because normally you would require either all or none of these options. Most of the cipher suites supported by Orbix support all of these association options, although there are a couple of integrity-only ciphers that do not support Confidentiality (see [Table 8 on page 125](#)). As a rule of thumb, if you want security you generally would want *all* of these association options.

Table 5: *Setting Quality of Protection Association Options*

Association Options	Client side—can appear in...	Server side—can appear in...
Confidentiality, Integrity, DetectReplay, and DetectMisordering	<code>policies:client_secure_invocation_policy:supports</code> <code>policies:client_secure_invocation_policy:requires</code>	<code>policies:target_secure_invocation_policy:supports</code> <code>policies:target_secure_invocation_policy:requires</code>

A typical secure application would list *all* of these association options in *all* of the configuration variables shown in [Table 5](#).

Note: Some of the sample configurations appearing in the generated configuration file require `Confidentiality`, but not the other qualities of protection. In practice, however, the list of required association options is implicitly extended to include the other qualities of protection, because the cipher suites that support `Confidentiality` also support the other qualities of protection. This is an example of where the security mechanism policy interacts with the secure invocation policies.

NoProtection

The `NoProtection` association option is used for two distinct purposes:

- *Disabling security selectively*—security is disabled, either in the client role or in the server role, if `NoProtection` appears as the sole *required* association option and as the sole *supported* association option in a secure invocation policy. This mechanism is selective in the sense that the client role and the server role can be independently configured as either secure or insecure.

Note: In this case, the `orb_plugins` configuration variable should include the `iiop` plug-in to enable insecure communication.

- *Making an application semi-secure*—an application is semi-secure, either in the client role or in the server role, if `NoProtection` appears as the sole *required* association option and as a *supported* association option along with other secure association options. The meaning of semi-secure in this context is, as follows:
 - ◆ *Semi-secure client*—the client will open either a secure or an insecure connection, depending on the disposition of the server (that is, depending on whether the server accepts only secure connections or only insecure connections). If the server is semi-secure, the type of connection opened depends on the order of the bindings in the `binding:client_binding_list`.

- ◆ *Semi-secure server*—the server accepts connections either from a secure or an insecure client.

Note: In this case, the `orb_plugins` configuration variable should include both the `iiop_tls` plug-in and the `iiop` plug-in.

Table 6 shows the configuration variables in which the `NoProtection` association option can appear.

Table 6: *Setting the NoProtection Association Option*

Association Option	Client side—can appear in...	Server side—can appear in...
NoProtection	<p><code>policies:client_secure_invocation_policy:supports</code></p> <p><code>policies:client_secure_invocation_policy:requires</code></p>	<p><code>policies:target_secure_invocation_policy:supports</code></p> <p><code>policies:target_secure_invocation_policy:requires</code></p>

References

For more information about setting association options, see the following:

- [“Securing Communications with SSL/TLS” on page 36.](#)
- The `demons.tls` scope in a generated Orbix configuration file.

Specifying Cipher Suites

Overview

This section explains how to specify the list of cipher suites that are made available to an application (client or server) for the purpose of establishing secure associations. During a security handshake, the client chooses a cipher suite that matches one of the cipher suites available to the server. The cipher suite then determines the security algorithms that are used for the secure association.

In this section

This section contains the following subsections:

Supported Cipher Suites	page 120
Setting the Mechanism Policy	page 126
Constraints Imposed on Cipher Suites	page 124

Supported Cipher Suites

Orbix cipher suites

The following cipher suites are supported by Orbix:

```
TLS_RSA_WITH_NULL_MD5
TLS_RSA_WITH_NULL_SHA
TLS_RSA_EXPORT_WITH_RC4_40_MD5
TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_DSS_WITH_AES_128_GCM_SHA256
TLS_DHE_DSS_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_RC4_128_SHA
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_AES_256_GCM_SHA384
TLS_AES_128_GCM_SHA256
TLS_CHACHA20_POLY1305_SHA256
```

Security algorithms

Each cipher suite specifies a set of three security algorithms, which are used at various stages during the lifetime of a secure association:

- *Key exchange algorithm*—used during the security handshake to enable authentication and the exchange of a symmetric key for subsequent communication. Must be a public key algorithm.
 - *Encryption algorithm*—used for the encryption of messages after the secure association has been established. Must be a symmetric (private key) encryption algorithm.
 - *Secure hash algorithm*—used for generating digital signatures. This algorithm is needed to guarantee message integrity.
-

Key exchange algorithms

The following key exchange algorithms are supported by Orbix:

DHE	Diffie-Hellman public key exchange
ECDHE	Elliptic Curve Diffie-Hellman public key exchange
ECDSA	Elliptic Curve Digital Signature Algorithm
RSA	Rivest Shamir Adleman (RSA) public key encryption using X.509v3 certificates. No restriction on the key size.
RSA_EXPORT	RSA public key encryption using X.509v3 certificates. Key size restricted to 512 bits.

Encryption algorithms

The following encryption algorithms are supported by Orbix:

RC4_40	A symmetric encryption algorithm developed by RSA data security. Key size restricted to 40 bits.
RC4_128	RC4 with a 128-bit key.
DES40_CBC	Data encryption standard (DES) symmetric encryption. Key size restricted to 40 bits.
DES_CBC	DES with a 56-bit key.
3DES_EDE_CBC	Triple DES (encrypt, decrypt, encrypt) with an effective key size of 168 bits.
AES_128_CBC	An AES (Advanced Encryption Standard) block cipher, with a 128-bit key, used for cipher block chaining.

AES_128_GCM	As AES_128, but using Galois Counter Mode to provide authentication.
AES_256_CBC	An AES block cipher, with a 256-bit key, used for cipher block chaining.
AES_256_GCM	As AES_256, but using Galois Counter Mode to provide authentication.

Secure hash algorithms

The following secure hash algorithms are supported by Orbix:

MD5	Message Digest 5 (MD5) hash algorithm. This algorithm produces a 128-bit digest.
SHA	Secure hash algorithm (SHA). This algorithm produces a 160-bit digest, but is somewhat slower than MD5.
SHA-256	A development of the SHA algorithm that produces a 256-bit digest.
SHA-384	A truncated version of SHA-512 that produces a 384-bit digest.

Cipher suite definitions

The Orbix cipher suites are defined as follows:

Table 7: *Cipher Suite Definitions*

Cipher	Protocol first enabled	Key Exchange	Auth.	Encryption	Mac / Hash
RSA_WITH_NULL_MD5	SSLv3	Kx=RSA	Au=RSA	Enc=NULL	Mac=MD5
RSA_WITH_NULL_SHA	SSLv3	Kx=RSA	Au=RSA	Enc=NULL	Mac=SHA1
RSA_EXPORT_WITH_RC4_40_MD5	SSLv3	Kx=RSA_EXPORT	Au=RSA	Enc=RC4 (40)	Mac=MD5
RSA_WITH_RC4_128_MD5	SSLv3	Kx=RSA	Au=RSA	Enc=RC4 (128)	Mac=MD5
RSA_WITH_RC4_128_SHA	SSLv3	Kx=RSA	Au=RSA	Enc=RC4 (128)	Mac=SHA1

Table 7: Cipher Suite Definitions

Cipher	Protocol first enabled	Key Exchange	Auth.	Encryption	Mac / Hash
RSA_EXPORT_WITH_RC2_CBC_40_MD5	SSLv3	Kx=RSA_EXPORT	Au=RSA	Enc=RC2 (40)	Mac=MD5
RSA_WITH_DES_CBC_SHA	SSLv3	Kx=RSA	Au=RSA	Enc=DES (56)	Mac=SHA1
RSA_WITH_3DES_EDE_CBC_SHA	SSLv3	Kx=RSA	Au=RSA	Enc=3DES (168)	Mac=SHA1
RSA_WITH_AES_128_CBC_SHA	SSLv3	Kx=RSA	Au=RSA	Enc=AES (128)	Mac=SHA1
RSA_WITH_AES_256_CBC_SHA	SSLv3	Kx=RSA	Au=RSA	Enc=AES (256)	Mac=SHA1
RSA_WITH_AES_128_CBC_SHA256	TLSv1.2	Kx=RSA	Au=RSA	Enc=AES (128)	Mac=SHA256
RSA_WITH_AES_256_CBC_SHA256	TLSv1.2	Kx=RSA	Au=RSA	Enc=AES (256)	Mac=SHA256
DHE_DSS_WITH_AES_128_CBC_SHA256	TLSv1.2	Kx=DH	Au=DSS	Enc=AES (128)	Mac=SHA256
DHE_RSA_WITH_AES_128_CBC_SHA256	TLSv1.2	Kx=RSA	Au=RSA	Enc=AES (128)	Mac=SHA256
DHE_DSS_WITH_AES_256_CBC_SHA256	TLSv1.2	Kx=DH	Au=DSS	Enc=AES (256)	Mac=SHA256
DHE_RSA_WITH_AES_256_CBC_SHA256	TLSv1.2	Kx=DH	Au=RSA	Enc=AES (256)	Mac=SHA256
RSA_WITH_AES_128_GCM_SHA256	TLSv1.2	Kx=RSA	Au=RSA	Enc=AESGCM(128)	Mac=SHA256
RSA_WITH_AES_256_GCM_SHA384	TLSv1.2	Kx=RSA	Au=RSA	Enc=AESGCM(256)	Mac=AEAD

Table 7: *Cipher Suite Definitions*

Cipher	Protocol first enabled	Key Exchange	Auth.	Encryption	Mac / Hash
DHE_RSA_WITH_AES_128_GCM_SHA256	TLSv1.2	Kx=DH	Au=RSA	Enc=AESGCM(128)	Mac=AEAD
DHE_RSA_WITH_AES_256_GCM_SHA384	TLSv1.2	Kx=DH	Au=RSA	Enc=AESGCM(256)	Mac=AEAD
DHE_DSS_WITH_AES_128_GCM_SHA256	TLSv1.2	Kx=DH	Au=DSS	Enc=AESGCM(128)	Mac=AEAD
DHE_DSS_WITH_AES_256_GCM_SHA384	TLSv1.2	Kx=any	Au=DSS	Enc=AESGCM(256)	Mac=AEAD
TLS_AES_128_GCM_SHA256	TLSv1.3	Kx=any	Au=any	Enc=AESGCM(128)	Mac=SHA256
TLS_AES_256_GCM_SHA384	TLSv1.3	Kx=any	Au=any	Enc=AESGCM(256)	Mac=SHA384
TLS_CHACHA20_POLY1305_SHA256	TLSv1.3	Kx=any	Au=any	Enc=CHACHA20/ POLY1305 (256)	Mac=SHA256
ECDHE_ECDSA_WITH_RC4_128_SHA	SSLv3	Kx=ECDH	Au=ECDSA	Enc=RC4 (128)	Mac=SHA1
ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	SSLv3	Kx=ECDH	Au=ECDSA	Enc=3DES (168)	Mac=SHA1
ECDHE_ECDSA_WITH_AES_128_CBC_SHA	SSLv3	Kx=ECDH	Au=ECDSA	Enc=AES (128)	Mac=SHA1
ECDHE_ECDSA_WITH_AES_256_CBC_SHA	SSLv3	Kx=ECDH	Au=ECDSA	Enc=AES (256)	Mac=SHA1
ECDHE_RSA_WITH_RC4_128_SHA	SSLv3	Kx=ECDH	Au=RSA	Enc=RC4 (128)	Mac=SHA1
ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	SSLv3	Kx=ECDH	Au=RSA	Enc=3DES (168)	Mac=SHA1

Table 7: *Cipher Suite Definitions*

Cipher	Protocol first enabled	Key Exchange	Auth.	Encryption	Mac / Hash
ECDHE_RSA_WITH_AES_128_CBC_SHA	SSLv3	Kx=ECDH	Au=RSA	Enc=AES (128)	Mac=SHA1
ECDHE_RSA_WITH_AES_256_CBC_SHA	SSLv3	Kx=ECDH	Au=RSA	Enc=AES (256)	Mac=SHA1
ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLSv1.2	Kx=ECDH	Au=ECDSA	Enc=AES (128)	Mac=SHA256
ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLSv1.2	Kx=ECDH	Au=ECDSA	Enc=AESGCM(256)	Mac=AED
ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLSv1.2	Kx=ECDH	Au=RSA	Enc=AES (128)	Mac=SHA256
ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLSv1.2	Kx=ECDH	Au=RSA	Enc=AES (256)	Mac=SHA384
ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLSv1.2	Kx=ECDH	Au=ECDSA	Enc=AESGCM(128)	Mac=AEAD
ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLSv1.2	Kx=ECDH	Au=ECDSA	Enc=AES (256)	Mac=SHA384
ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLSv1.2	Kx=ECDH	Au=RSA	Enc=AESGCM(128)	Mac=AEAD
ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLSv1.2	Kx=ECDH	Au=RSA	Enc=AESGCM(256)	Mac=AEAD

Reference

For further details about cipher suites in the context of TLS, see RFC 2246 from the Internet Engineering Task Force (IETF). This document is available from the IETF Web site: <http://www.ietf.org>.

Setting the Mechanism Policy

Mechanism policy

To specify cipher suites, use the *mechanism policy*. The mechanism policy is a client and server side security policy that determines

- Which TLS protocol is used, and
 - Which specific cipher suites are to be used.
-

The `protocol_version` configuration variable

You can specify which TLS protocol versions are used with a transport protocol by assigning a list of protocol versions to the `policies:iiop_tls:mechanism_policy:protocol_version` configuration variable for IIOP/TLS and the `policies:https:mechanism_policy:protocol_version` configuration variable for HTTPS. For example:

```
# Orbix Configuration File
policies:iiop_tls:mechanism_policy:protocol_version = ["TLS_V1",
"TLS_V1_2"];
```

You can set the `protocol_version` configuration variable to include one or more of the following protocols:

```
TLS_V1_3
TLS_V1_2
TLS_V1_1
TLS_V1
SSL_V3
```

And a special setting for interoperating with an application deployed on the z/OS platform (to work around a bug in IBM's System/SSL toolkit):

```
SSL_V2V3
```

This special `SSL_V2V3` setting is not required for z/OS 1.5 or higher.

The order of the entries in the `protocol_version` list is unimportant. During the SSL/TLS handshake, the highest common protocol will be negotiated.

Notes when using TLS v1.3

Be aware of the following when using TLS v1.3:

- TLS v1.3 supports only these cipher suites:
 - `TLS_AES_128_GCM_SHA256`

- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256

Be sure to configure at least one of these cipher suites when using TLS v1.3.

- If TLS v1.3 is configured as part of a range of protocols but no TLS v1.3 supported cipher suites are configured, TLS v1.3 is effectively "turned off" and will not be used.
- If TLS v1.3 is configured as part of a range of protocols, be sure to configure non-TLS v1.3 cipher suites in addition to the TLS v1.3 cipher suites. A handshake failure can occur if the correct set of cipher suites is not configured.
- TLS v1.3 has the following certificate requirements:
 - An RSA key size of 2048 or greater
 - An ECC key size of 256 or greater
- Ensure you have the correct RACF authority to use a certificate with an RSA key size of 2048 or greater with TLS v1.3. If not, handshake failures can occur.
- If using a certificate with an ECC key, signed by a Certificate Authority that uses an RSA key, handshake failures can occur when using protocol TLS v1.1 or older, as SystemSSL will want to negotiate cipher suites that are not supported by Orbix Mainframe. A certificate with an ECC key signed by a Certificate Authority that also uses an ECC key will not encounter this issue with TLS v1.1 or older.

The cipher suites configuration variable

You can specify the cipher suites available to a transport protocol by setting the `policies:iiop_tls:mechanism_policy:ciphersuites` configuration variable for IIOp/TLS and the `policies:https:mechanism_policy:ciphersuites` configuration variable for HTTPS. For example:

```
# Orbix Configuration File
policies:iiop_tls:mechanism_policy:ciphersuites =
["TLS_AES_256_GCM_SHA384","RSA_WITH_AES_256_CBC_SHA256",
"RSA_WITH_AES_256_CBC_SHA", "RSA_WITH_AES_128_CBC_SHA",
"RSA_WITH_AES_128_CBC_SHA256"];
```

Cipher suite order

The order of the entries in the mechanism policy's cipher suites list is important.

During a security handshake, the client sends a list of acceptable cipher suites to the server. The server then chooses the first of these cipher suites that it finds acceptable. The secure association is, therefore, more likely to use those cipher suites that are near the beginning of the `ciphersuites` list.

Valid cipher suites

You can specify any of the following cipher suites:

```

RSA_WITH_NULL_MD5
RSA_WITH_NULL_SHA
RSA_EXPORT_WITH_RC4_40_MD5
RSA_WITH_RC4_128_MD5
RSA_WITH_RC4_128_SHA
RSA_WITH_DES_CBC_SHA
RSA_WITH_3DES_EDE_CBC_SHA
RSA_WITH_AES_128_CBC_SHA
RSA_WITH_AES_256_CBC_SHA
RSA_WITH_AES_128_CBC_SHA256
RSA_WITH_AES_256_CBC_SHA256
DHE_DSS_WITH_AES_128_CBC_SHA256
DHE_RSA_WITH_AES_128_CBC_SHA256
DHE_DSS_WITH_AES_256_CBC_SHA256
DHE_RSA_WITH_AES_256_CBC_SHA256
RSA_WITH_AES_128_GCM_SHA256
RSA_WITH_AES_256_GCM_SHA384
DHE_RSA_WITH_AES_128_GCM_SHA256
DHE_RSA_WITH_AES_256_GCM_SHA384
DHE_DSS_WITH_AES_128_GCM_SHA256
DHE_DSS_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_RC4_128_SHA
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384

```

```
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_AES_256_GCM_SHA384
TLS_AES_128_GCM_SHA256
TLS_CHACHA20_POLY1305_SHA256
```

Default values

If no cipher suites are specified through configuration or application code, the following apply:

```
TLS_AES_256_GCM_SHA384
RSA_WITH_AES_256_CBC_SHA256
RSA_WITH_AES_256_CBC_SHA
RSA_WITH_AES_128_CBC_SHA
RSA_WITH_AES_128_CBC_SHA256
```


Configuring SSL/TLS Authentication

This chapter describes how to configure the authentication requirements for your application.

In this chapter

This chapter discusses the following topics:

Requiring Authentication	page 132
Specifying an Application's Own Certificate	page 139
Advanced Configuration Options	page 142

Requiring Authentication

Overview

This section discusses how to specify whether a target object must authenticate itself to a client and whether the client must authenticate itself to the target. For a given client-server link, the authentication requirements are governed by the following policies:

- Client secure invocation policy.
- Target secure invocation policy.
- Mechanism policy.

These policies are explained in detail in [“Configuring SSL/TLS Secure Associations” on page 103](#). This section focuses only on those aspects of the policies that affect authentication.

In this section

There are two possible arrangements for a TLS secure association:

Target Authentication Only	page 133
Target and Client Authentication	page 136

Target Authentication Only

Overview

When an application is configured for target authentication only, the target authenticates itself to the client but the client is not authenticated to the target object—see [Figure 12](#).

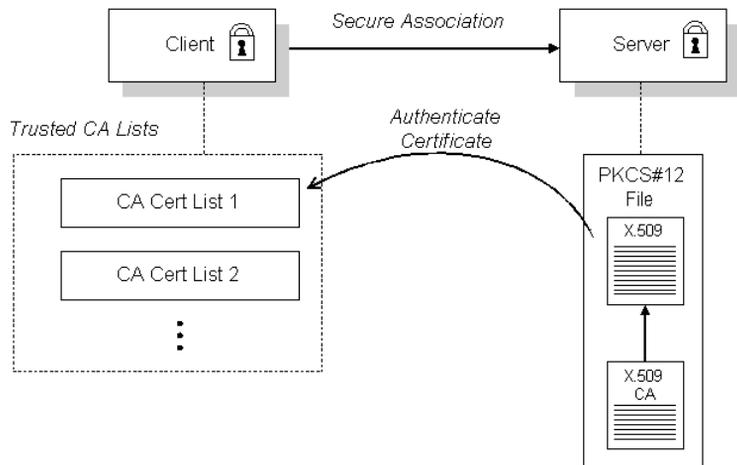


Figure 12: Target Authentication Only

Security handshake

Prior to running the application, the client and server should be set up as follows:

- A certificate chain is associated with the server—the certificate chain is provided in the form of a PKCS#12 file. See [“Specifying an Application’s Own Certificate”](#) on page 139.
- One or more lists of trusted certification authorities (CA) are made available to the client.

During the security handshake, the server sends its certificate chain to the client—see [Figure 12](#). The client then searches its trusted CA lists to find a CA certificate that matches one of the CA certificates in the server’s certificate chain.

Client configuration

For target authentication only, the client policies should be configured as follows:

- Client secure invocation policy—must be configured both to *require* and *support* the `EstablishTrustInTarget` association option.
- Mechanism policy—at least one of the specified cipher suites must be capable of supporting target authentication. All of the cipher suites currently provided by Orbix support target authentication.

Server configuration

For target authentication only, the target policies should be configured as follows:

- Target secure invocation policy—must be configured to *support* the `EstablishTrustInTarget` association option.
- Mechanism policy—at least one of the specified cipher suites must be capable of supporting target authentication. All of the cipher suites currently provided by Orbix support target authentication.

Example of target authentication only

The following sample extract from an Orbix configuration file shows a configuration for a CORBA client application, `bank_client`, and a CORBA server application, `bank_server`, in the case of target authentication only.

```
# Orbix Configuration File
...
policies:iiop_tls:mechanism_policy:protocol_version =
    "TLS_V1_2";
policies:mechanism_policy:ciphersuites =
    ["RSA_WITH_RC4_128_SHA",
     "RSA_WITH_AES_256_CBC_SHA256",
     "RSA_WITH_RC4_128_MD5",
     "RSA_WITH_AES_256_CBC_SHA",
     "RSA_WITH_DES_CBC_SHA",
     "RSA_WITH_3DES_EDE_CBC_SHA",
     "RSA_WITH_AES_128_CBC_SHA",
     "RSA_WITH_AES_128_CBC_SHA256"];

bank_server {
    policies:iiop_tls:target_secure_invocation_policy:requires =
        ["Confidentiality"];
    policies:iiop_tls:target_secure_invocation_policy:supports =
        ["Confidentiality", "Integrity", "DetectReplay",
         "DetectMisordering", "EstablishTrustInTarget"];
    ...
};

bank_client {
    ...
    policies:iiop_tls:client_secure_invocation_policy:requires =
        ["Confidentiality", "EstablishTrustInTarget"];
    policies:iiop_tls:client_secure_invocation_policy:supports =
        ["Confidentiality", "Integrity", "DetectReplay",
         "DetectMisordering", "EstablishTrustInTarget"];
};
```

Target and Client Authentication

Overview

When an application is configured for target and client authentication, the target authenticates itself to the client and the client authenticates itself to the target. This scenario is illustrated in [Figure 13](#). In this case, the server and the client each require an X.509 certificate for the security handshake.

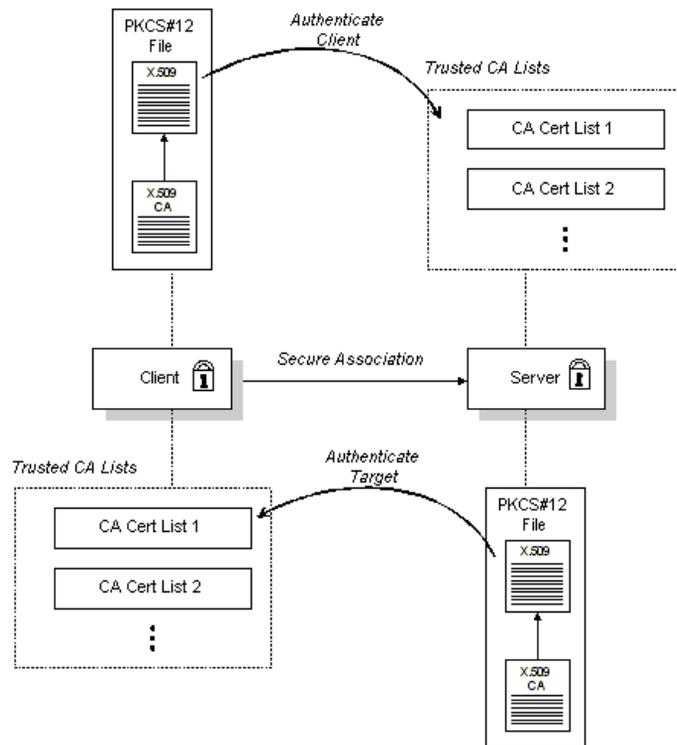


Figure 13: Target and Client Authentication

Security handshake

Prior to running the application, the client and server should be set up as follows:

- Both client and server have an associated certificate chain (PKCS#12 file)—see [“Specifying an Application’s Own Certificate” on page 139](#).
- Both client and server are configured with lists of trusted certification authorities (CA).

During the security handshake, the server sends its certificate chain to the client, and the client sends its certificate chain to the server—see [Figure 12](#).

Client configuration

For target and client authentication, the client policies should be configured as follows:

- Client secure invocation policy—must be configured both to *require* and *support* the `EstablishTrustInTarget` association option. The client also must *support* the `EstablishTrustInClient` association option.
 - Mechanism policy—at least one of the specified cipher suites must be capable of supporting target authentication.
-

Server configuration

For target and client authentication, the target policies should be configured as follows:

- Target secure invocation policy—must be configured to *support* the `EstablishTrustInTarget` association option. The target must also *require* and *support* the `EstablishTrustInClient` association option.
- Mechanism policy—at least one of the specified cipher suites must be capable of supporting target and client authentication.

Example of target and client authentication

The following sample extract from an Orbix configuration file shows a configuration for a client application, `secure_client_with_cert`, and a server application, `secure_server_enforce_client_auth`, in the case of target and client authentication.

```
# Orbix Configuration File
...
policies:iiop_tls:mechanism_policy:protocol_version =
    "TLS_V1_2";
policies:mechanism_policy:ciphersuites =
    ["RSA_WITH_RC4_128_SHA",
     "RSA_WITH_AES_256_CBC_SHA256",
     "RSA_WITH_RC4_128_MD5",
     "RSA_WITH_AES_256_CBC_SHA",
     "RSA_WITH_DES_CBC_SHA",
     "RSA_WITH_3DES_EDE_CBC_SHA",
     "RSA_WITH_AES_128_CBC_SHA",
     "RSA_WITH_AES_128_CBC_SHA256"];

secure_server_enforce_client_auth
{
    policies:iiop_tls:target_secure_invocation_policy:requires =
        ["EstablishTrustInClient", "Confidentiality"];
    policies:iiop_tls:target_secure_invocation_policy:supports =
        ["EstablishTrustInClient", "Confidentiality", "Integrity",
         "DetectReplay", "DetectMisordering",
         "EstablishTrustInTarget"];
    ...
};

secure_client_with_cert
{
    policies:iiop_tls:client_secure_invocation_policy:requires =
        ["Confidentiality", "EstablishTrustInTarget"];
    policies:iiop_tls:client_secure_invocation_policy:supports =
        ["Confidentiality", "Integrity", "DetectReplay",
         "DetectMisordering", "EstablishTrustInClient",
         "EstablishTrustInTarget"];
    ...
};
```

Specifying an Application's Own Certificate

Overview

To enable an Orbix application to identify itself, it must be associated with an X.509 certificate. The X.509 certificate is needed during an SSL/TLS handshake, where it is used to authenticate the application to its peers. The method you use to specify the certificate depends on the type of application:

- *Security unaware*—configuration only,
- *Security aware*—configuration or programming.

This section describes how to specify a certificate by configuration only. For details of the programming approach, see [“Authentication” on page 209](#).

PKCS#12 files

In practice, the TLS protocol needs more than just an X.509 certificate to support application authentication. Orbix therefore stores X.509 certificates in a PKCS#12 file, which contains the following elements:

- The application certificate, in X.509 format.
- One or more certificate authority (CA) certificates, which vouch for the authenticity of the application certificate (see also [“Certification Authorities” on page 82](#)).
- The application certificate's private key (encrypted).

In addition to the encryption of the private key within the certificate, the whole PKCS#12 certificate is also stored in encrypted form.

Note: The same pass phrase is used both for the encryption of the private key within the PKCS#12 file and for the encryption of the PKCS#12 file overall. This condition (same pass phrase) is not officially part of the PKCS#12 standard, but it is enforced by most Web browsers and by Orbix.

Figure 14 shows the typical elements in a PKCS#12 file.

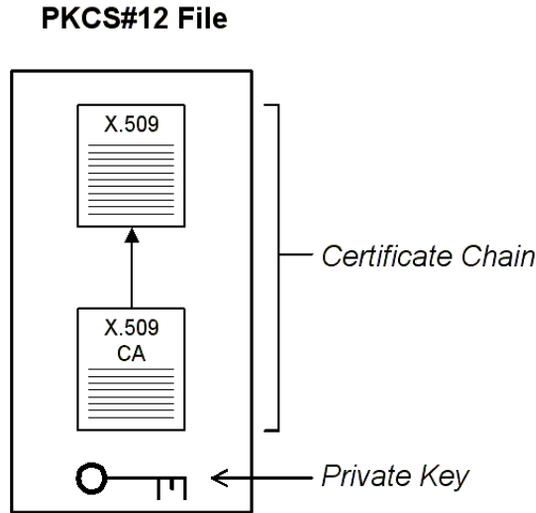


Figure 14: Elements in a PKCS#12 File

SSL/TLS principal sponsor

The SSL/TLS principal sponsor is a piece of code embedded in the security plug-in that obtains SSL/TLS authentication information for an application. It is configured by setting variables in the Orbix configuration.

Single or multiple certificates

The SSL/TLS principal sponsor is limited to specifying a *single* certificate for each ORB scope. This is sufficient for most applications.

Specifying multiple certificates for a single ORB can only be achieved by programming (see [“Authentication” on page 209](#)). If an application is programmed to own multiple certificates, that application ought to be accompanied by documentation that explains how to specify the certificates.

Credentials sharing

Normally, when you specify an own credential using the SSL/TLS principal sponsor, the credential is available only to the ORB that created it. By setting the `plugins:security:share_credentials_across_orbs` variable to `true`, however, the own credentials created by one ORB are automatically made available to any other ORBs that are configured to share credentials.

Specifying the HFS database or RACF key ring

Before setting the principal sponsor configuration variables on z/OS, you must also indicate the name of a HFS key database or an RACF key ring to use. See [“Specifying the Source of Certificates for an z/OS Application” on page 95](#).

Principal sponsor configuration

To use a principal sponsor, set the `principal_sponsor` configuration variables, as follows:

1. Set the variable `principal_sponsor:use_principal_sponsor` to `true`.
 2. Provide values for the `principal_sponsor:auth_method_id` and `principal_sponsor:auth_method_data` variables.
-

Example configuration

For example, to use a certificate labelled `bank_server`, (as used in [“Importing Certificates from Another Platform into RACF” on page 89](#)) set the `principal_sponsor` configuration variables as follows:

```
principal_sponsor:use_principal_sponsor = "true";  
principal_sponsor:auth_method_id = "security_label";  
principal_sponsor:auth_method_data = ["label=bank_server"];
```

The `principal_sponsor:auth_method_id` configuration variable indicates the source that Orbix should use to get the certificate. In this case the `security_label` value indicates a label in a key ring.

Advanced Configuration Options

Overview

For added security, Orbix allows you to apply extra conditions on certificates. Before reading this section you might find it helpful to consult [“Managing Certificates” on page 79](#), which provides some background information on the structure of certificates.

In this section

This section discusses the following advanced configuration options:

Setting a Maximum Certificate Chain Length	page 143
Applying Constraints to Certificates	page 144

Setting a Maximum Certificate Chain Length

Max chain length policy

You can use the `MaxChainLengthPolicy` to enforce the maximum length of certificate chains presented by a peer during handshaking.

A certificate chain is made up of a root CA at the top, an application certificate at the bottom and any number of CA intermediaries in between. The length that this policy applies to is the (inclusive) length of the chain from the application certificate presented to the first signer in the chain that appears in the list of trusted CA's (as specified in the `TrustedCAListPolicy`).

Example

For example, a chain length of 2 mandates that the certificate of the immediate signer of the peer application certificate presented must appear in the list of trusted CA certificates.

Configuration variable

You can specify the maximum length of certificate chains used in `MaxChainLengthPolicy` with the `policies:iiop_tls:max_chain_length_policy` and `policies:https:max_chain_length_policy` configuration variables. For example:

```
policies:iiop_tls:max_chain_length_policy = "4";
```

Note: The `max_chain_length_policy` is not currently supported on the z/OS platform.

Default value

The default value is 2 (that is, the application certificate and its signer, where the signer must appear in the list of trusted CA's).

Applying Constraints to Certificates

Certificate constraints policy

You can use the `CertConstraintsPolicy` to apply constraints to peer X.509 certificates by the default `CertificateValidatorPolicy`. These conditions are applied to the owner's distinguished name (DN) on the first certificate (peer certificate) of the received certificate chain. Distinguished names are made up of a number of distinct fields, the most common being Organization Unit (OU) and Common Name (CN).

Configuration variable

You can specify a list of constraints to be used by `CertConstraintsPolicy` through the `policies:iiop_tls:certificate_constraints_policy` or `policies:https:certificate_constraints_policy` configuration variables. For example:

```
policies:iiop_tls:certificate_constraints_policy =
    ["CN=Johnny*,OU=[unit1|IT_SSL],O=Orbix
    Services,C=Ireland,ST=Dublin,L=Earth", "CN=Paul*,OU=SSLTEAM,O=
    Orbix Services,C=Ireland,ST=Dublin,L=Earth",
    "CN=TheOmnipotentOne"];
```

Constraint language

These are the special characters and their meanings in the constraint list:

*	Matches any text. For example: an* matches ant and anger, but not aunt
[]	Grouping symbols.
	Choice symbol. For example: OU=[unit1 IT_SSL] signifies that if the OU is unit1 or IT_SSL, the certificate is acceptable.
=, !=	Signify equality and inequality respectively.

Example

This is an example list of constraints:

```
policies:iiop_tls:certificate_constraints_policy = [
    "OU=[unit1|IT_SSL],CN=Steve*,L=Dublin",
    "OU=IT_ART*,OU!=IT_ARTtesters,CN=[Jan|Donal],ST=
    Boston" ];
```

This constraint list specifies that a certificate is deemed acceptable if and only if it satisfies one or more of the constraint patterns:

```

If
  The OU is unit1 or IT_SSL
  And
  The CN begins with the text Steve
  And
  The location is Dublin
Then the certificate is acceptable
Else (moving on to the second constraint)
If
  The OU begins with the text IT_ART but isn't IT_ARTtesters
  And
  The common name is either Donal or Jan
  And
  The State is Boston
Then the certificate is acceptable
Otherwise the certificate is unacceptable.

```

The language is like a boolean OR, trying the constraints defined in each line until the certificate satisfies one of the constraints. Only if the certificate fails all constraints is the certificate deemed invalid.

Note that this setting can be sensitive about white space used within it. For example, "CN =" might not be recognized, where "CN=" is recognized.

Distinguished names

For more information on distinguished names, see [“ASN.1 and Distinguished Names” on page 307](#).

Part 4

CSlv2 Administration

In this part

This part contains the following chapters:

Introduction to CSlv2	page 149
Configuring CSlv2 Authentication over Transport	page 157
Configuring CSlv2 Identity Assertion	page 177

Introduction to CSlv2

CSlv2 is the OMG's Common Secure Interoperability protocol v2.0, which can provide the basis for application-level security in CORBA applications. The Orbix Security Framework uses CSlv2 to transmit usernames and passwords, and asserted identities between applications.

In this chapter

This chapter discusses the following topics:

CSlv2 Features	page 150
Basic CSlv2 Scenarios	page 152

CSiv2 Features

Overview

This section gives a quick overview of the basic features provided by CSiv2 application-level security. Fundamentally, CSiv2 is a general, interoperable mechanism for propagating security data between applications. Because CSiv2 is designed to complement SSL/TLS security, CSiv2 focuses on providing security features not covered by SSL/TLS.

Application-level security

CSiv2 is said to provide *application-level security* because, in contrast to SSL/TLS, security data is transmitted above the transport layer and the security data is sent after a connection has been established.

Transmitting CSiv2-related security data

The CSiv2 specification defines a new GIOP service context type, the *security attribute service context*, which is used to transmit CSiv2-related security data. There are two important specializations of GIOP:

- IIOp—the Internet inter-ORB protocol, which specialises GIOP to the TCP/IP transport, is used to send CSiv2 data between CORBA applications.
 - RMI/IIOP—RMI over IIOP, which is an IIOP-compatible version of Java's Remote Method Invocation (RMI) technology, is used to send CSiv2 data between EJB applications and also for CORBA-to-EJB interoperability.
-

CSiv2 mechanisms

The following CSiv2 mechanisms are supported:

- [CSiv2 authentication over transport mechanism](#).
 - [CSiv2 identity assertion mechanism](#).
-

CSiv2 authentication over transport mechanism

The CSiv2 authentication over transport mechanism provides a simple client authentication mechanism, based on a username and a password. This mechanism propagates a username, password, and domain name to the server. The server then authenticates the username and password before allowing the invocation to proceed.

CSlv2 identity assertion mechanism

The CSlv2 identity assertion mechanism provides a way of asserting the identity of a caller without performing authentication. This mechanism is usually used to propagate a caller identity that has already been authenticated at an earlier point in the system.

Applicability of CSlv2

CSlv2 is applicable to both CORBA technology. CSlv2 can be used by the following kinds of application:

- CORBA C++ applications.
- CORBA Java applications.

Basic CSIV2 Scenarios

Overview

The CSIV2 specification provides two independent mechanisms for sending credentials over the transport (authentication over transport, and identity assertion), but the CSIV2 specification does not mandate how the transmitted credentials are used. Hence, there are many different ways of using CSIV2 and different ways to integrate it into a security framework (such as iSF).

This section describes some of the basic scenarios that illustrate typical CSIV2 usage.

In this section

This section contains the following subsections:

CSIV2 Authentication over Transport Scenario	page 153
CSIV2 Identity Assertion Scenario	page 154

CSIV2 Authentication over Transport Scenario

Overview

Figure 15 shows a basic CSIV2 scenario where a CORBA client and a CORBA server are configured to use the CSIV2 authentication over transport mechanism.

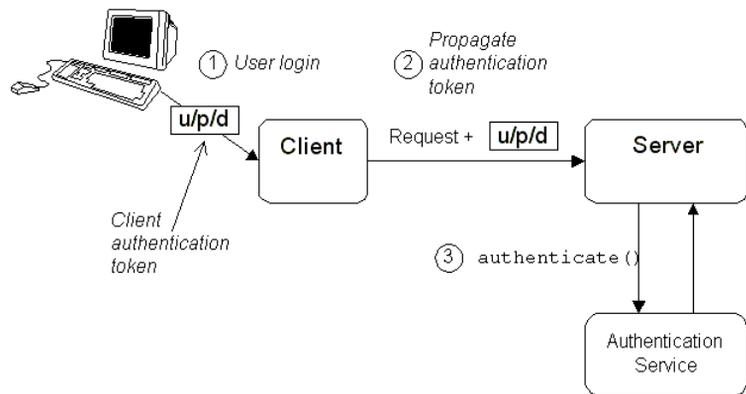


Figure 15: Basic CSIV2 Authentication over Transport Scenario

Scenario description

The scenario shown in Figure 15 can be described as follows:

Stage	Description
1	The user enters a username, password, domain name on the client side (user login).
2	When the client makes a remote invocation on the server, CSIV2 transmits the username/password/domain authentication data to the server in a security attribute service context.
3	The server authenticates the received username/password before allowing the invocation to proceed.

More details

For more details about authentication over transport, see [“Configuring CSIV2 Authentication over Transport” on page 157](#).

CSlv2 Identity Assertion Scenario

Overview

Figure 16 shows a basic CSlv2 scenario where a client and an intermediate server are configured to use the CSlv2 authentication over transport mechanism, and the intermediate server and a target server are configured to use the CSlv2 identity assertion mechanism. In this scenario, the client invokes on the intermediate server, which then invokes on the target server.

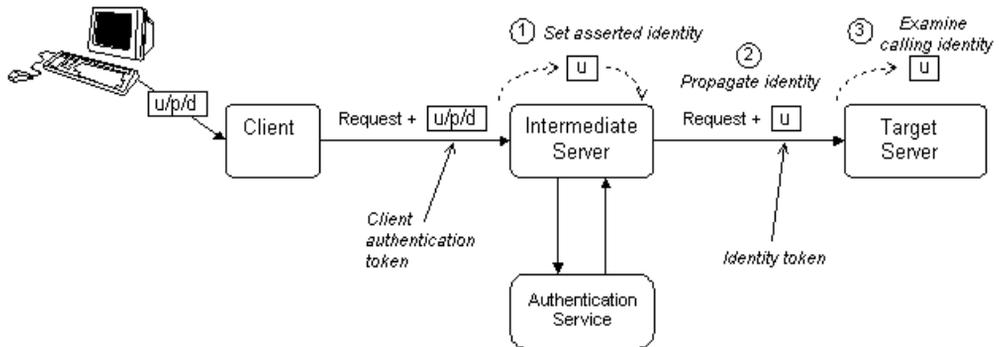


Figure 16: Basic CSlv2 Identity Assertion Scenario

Scenario description

The second stage of the scenario shown in Figure 16 (intermediate server invokes an operation on the target server) can be described as follows:

Stage	Description
1	<p>The intermediate server can set the identity that will be asserted to the target in one of two ways:</p> <ul style="list-style-type: none"> • Implicitly—if the execution context has an associated CSlv2 received credentials, the intermediate server extracts the user identity from the received credentials, or • Explicitly—by programming.
2	<p>When the intermediate server makes a remote invocation on the target server, CSlv2 transmits the user identity data to the server in a security attribute service context.</p>

Stage	Description
3	The target server can access the propagated user identity programmatically (by extracting it from a <code>SecurityLevel2::ReceivedCredentials</code> object).

More details

For more details about identity assertion, see [“Configuring CSiv2 Identity Assertion” on page 177](#).

Configuring CSlv2 Authentication over Transport

This chapter explains the concepts underlying the CSlv2 authentication over transport mechanism and provides details of how to configure a client and a server to use this mechanism.

In this chapter

This chapter discusses the following topics:

CSlv2 Authentication Scenario	page 158
SSL/TLS Prerequisites	page 162
Requiring CSlv2 Authentication	page 164
Providing an Authentication Service	page 167
Providing a Username and Password	page 168
Sample Configuration	page 172

CSIV2 Authentication Scenario

Overview

This section describes a typical CSIV2 authentication scenario, where the client is authenticated over the transport by providing a username and a password.

Authentication over transport

The CSIV2 *authentication over transport* mechanism is a simple client authentication mechanism based on a username and a password. In a system with a large number of clients, it is significantly easier to administer CSIV2 client authentication than it is to administer SSL/TLS client authentication.

CSIV2 authentication is said to be *over transport*, because the authentication step is performed at the General Inter-ORB Protocol (GIOP) layer. Specifically, authentication data is inserted into the service context of a GIOP request message. CSIV2 authentication, therefore, occurs *after* a connection has been established (in contrast to SSL/TLS authentication).

GSSUP mechanism

The Generic Security Service Username/Password (GSSUP) mechanism is the basic authentication mechanism supported by CSIV2 at Level 0 conformance. Currently, this is the only authentication mechanism supported by Orbix's implementation of CSIV2.

Dependency on SSL/TLS

Note, that CSIV2 authentication over transport *cannot provide adequate security on its own*. The authentication over transport mechanism relies on the transport layer security, that is SSL/TLS, to provide the following additional security features:

- Server authentication.
- Privacy of communication.
- Message integrity.

CSlv2 scenario

Figure 17 shows a typical scenario for CSlv2 authentication over transport:

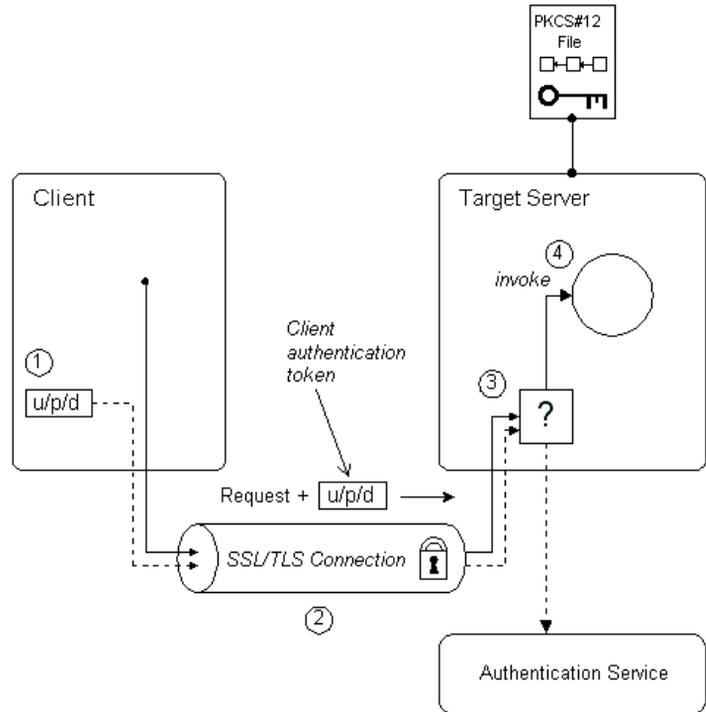


Figure 17: CSlv2 Authentication Over Transport Scenario

How CSlv2 authentication over transport proceeds

As shown in Figure 17 on page 159, the authentication over transport mechanism proceeds as follows:

Stage	Description
1	When a client initiates an operation invocation on the target, the client's CSI plug-in inserts a client authentication token (containing username/password/domain) into the GIOP request message.

Stage	Description
2	The request, together with the client authentication token, is sent over the SSL/TLS connection. The SSL/TLS connection provides privacy and message integrity, ensuring that the username and password cannot be read by eavesdroppers.
3	Before permitting the request to reach the target object, the CSI server interceptor calls an application-supplied object (the authentication service) to check the username/password combination.
4	If the username/password combination are authenticated successfully, the request is allowed to reach the target object; otherwise the request is blocked and an error returned to the client.

SSL/TLS connection

The client and server should both be configured to use a secure SSL/TLS connection. In this scenario, the SSL/TLS connection is configured for target authentication only.

See [“SSL/TLS Prerequisites” on page 162](#) for details of the SSL/TLS configuration for this scenario.

Client authentication token

A *client authentication token* contains the data that a client uses to authenticate itself to a server through the CSiv2 authentication over transport mechanism, as follows:

- *Username*—a UTF-8 character string, which is guaranteed not to undergo conversion when it is sent over the wire.
- *Password*—a UTF-8 character string, which is guaranteed not to undergo conversion when it is sent over the wire.
- *Domain*—a string that identifies the CSiv2 authentication domain within which the user is authenticated.

Note: The client’s domain should match the target domain, which is specified by the

```
policies:csi:auth_over_transport:server_domain_name
```

configuration variable on the server side.

The client authentication token is usually initialized by the *CSlv2 principal sponsor* (which prompts the user to enter the username/password and domain). See [“Providing a Username and Password” on page 168](#).

Authentication service

The *authentication service* is an external service that checks the username and password received from the client. If the authentication succeeds, the request is allowed to proceed and an invocation is made on the target object; if the authentication fails, the request is automatically blocked and a `CORBA::NO_PERMISSION` system exception is returned to the client.

See [“Providing an Authentication Service” on page 167](#).

SSL/TLS Prerequisites

Overview

The SSL/TLS protocol is an essential complement to CSiv2 security. The CSiv2 authentication over transport mechanism relies on SSL/TLS to provide the following additional security features:

- Server authentication.
- Privacy of communication.
- Message integrity.

WARNING: If you do not enable SSL/TLS for the client-server connection, the GSSUP username and password would be sent over the wire unencrypted and, therefore, could be read by eavesdroppers.

SSL/TLS target authentication only

For the scenario depicted in [Figure 17 on page 159](#), the SSL/TLS connection is configured for target authentication only. The SSL/TLS configuration can be summarized as follows:

- *Client-side SSL/TLS configuration*—the client requires confidentiality, message integrity, and the `EstablishTrustInTarget` SSL/TLS association option. No X.509 certificate is provided on the client side, because the client is not authenticated at the transport layer.
 - *Server-side SSL/TLS configuration*—the server requires confidentiality and message integrity, but the `EstablishTrustInClient` SSL/TLS association option is not required. An X.509 certificate is provided on the server side to enable the client to authenticate the server.
-

Configuration samples

The SSL/TLS configuration of this CSiv2 scenario is based on the following TLS demonstration configurations in your Orbix configuration (`DomainName.cfg` file or CFR service):

- `demos.tls.secure_client_with_no_cert`
- `demos.tls.secure_server_no_client_auth`

SSL/TLS principal sponsor configuration

In this scenario, the SSL/TLS principal sponsor needs to be enabled only on the server side, because it is only the server that has an associated X.509 certificate.

Note: The SSL/TLS principal sponsor is completely independent of the CSlv2 principal sponsor (see [“CSlv2 principal sponsor” on page 168](#)). It is possible, therefore, to enable both of the principal sponsors within the same application.

References

See [“Sample Configuration” on page 172](#) for a detailed example of the client and server SSL/TLS configuration.

See [“SSL/TLS Administration” on page 97](#) for complete details of configuring and administering SSL/TLS.

Requiring CSiv2 Authentication

Overview

This section describes the *minimal* configuration needed to enable CSiv2 authentication over transport. In a typical system, however, you also need to configure SSL/TLS (see [“SSL/TLS Prerequisites” on page 162](#)) and the CSiv2 principal sponsor (see [“Providing a Username and Password” on page 168](#)).

Loading the CSI plug-in

To enable CSiv2 for a C++ or Java application, you must include the `csi` plug-in in the `orb_plugins` list in your Orbix configuration. The `binding:client_binding_list` and `binding:server_binding_list` must also be initialized with the proper list of interceptor combinations.

Sample settings for these configuration variables can be found in the `demos.tls.csi2` configuration scope of your Orbix configuration. For example, you can load the `csi` plug-in with the following configuration:

```
# Orbix configuration file
csiv2 {
  orb_plugins = ["local_log_stream", "iiop_profile", "giop",
               "iiop_tls", "csi"];

  binding:client_binding_list = ["GIOP+EGMIOP",
                                "OTS+POA_Coloc", "POA_Coloc", "OTS+TLS_Coloc+POA_Coloc",
                                "TLS_Coloc+POA_Coloc", "GIOP+SHMIOP", "CSI+OTS+GIOP+IIOP",
                                "CSI+GIOP+IIOP", "CSI+OTS+GIOP+IIOP_TLS",
                                "CSI+GIOP+IIOP_TLS"];

  binding:server_binding_list = ["CSI"];
  ...
};
```

Client configuration

A client can be configured to support CSiv2 authentication over transport, as follows:

```
# Orbix configuration file
policies:csi:auth_over_transport:client_supports =
  ["EstablishTrustInClient"];
```

Client CSiv2 association options

The `EstablishTrustInClient` option is a CSiv2 association option. Including this option in the `policies:csi:auth_over_transport:client_supports` list indicates that the client supports the CSiv2 authentication over transport mechanism.

Server configuration

A server can be configured to support CSiv2 authentication over transport, as follows:

```
# Orbix configuration file
policies:csi:auth_over_transport:target_supports =
  ["EstablishTrustInClient"];
policies:csi:auth_over_transport:target_requires =
  ["EstablishTrustInClient"];
policies:csi:auth_over_transport:server_domain_name =
  "AuthDomain";
policies:csi:auth_over_transport:authentication_service =
  "csiv2.AuthenticationServiceObject";
```

Server CSiv2 association options

Including the `EstablishTrustInClient` CSiv2 association option in the `policies:csi:auth_over_transport:target_supports` list indicates that the server *supports* the CSiv2 authentication over transport mechanism. Including the `EstablishTrustInClient` CSiv2 association option in the `policies:csi:auth_over_transport:target_requires` list indicates that the server *requires* clients to authenticate themselves using the CSiv2 authentication over transport mechanism. If the client fails to authenticate itself to the server when the server requires it, the server throws a `CORBA::NO_PERMISSION` system exception back to the client.

Server domain name

The server domain name is the name of a valid CSiv2 authentication domain. A CSiv2 authentication domain is an administrative unit within which a username/password combination is authenticated. A CSiv2 client will check that the domain name in its CSiv2 credentials is the same as the domain name set on the server side by the `policies:csi:auth_over_transport:server_domain_name` configuration variable. If the domain in the client credentials is an empty string, however, the domain always matches (the empty string is treated as a wildcard).

Authentication service

The `authentication_service` variable specifies a Java class that provides an implementation of the authentication service. This enables you to provide a custom implementation of the CSlv2 authentication service in Java.

When using CSlv2 in the context of the Orbix Security Framework, however, this configuration variable should be omitted. In the Orbix Security Framework, the GSP plug-in specifies the CSlv2 authentication service programmatically.

See [“Providing an Authentication Service” on page 167](#) for more details.

Providing an Authentication Service

Overview

An implementation of the CSIV2 authentication service can be specified in one of the following ways:

- [By configuration \(Java only\)](#).
- [By programming a policy \(Java only\)](#).
- [By registering an initial reference](#).

By configuration (Java only)

In Java, the authentication service is provided by a customizable class which can be loaded by setting the

`policies:csi:auth_over_transport:authentication_service` configuration variable to the fully-scoped name of the Java class.

By programming a policy (Java only)

In Java, you can specify a CSIV2 authentication service object programmatically by setting the `IT_CSI::CSI_SERVER_AS_POLICY` policy with an `IT_CSI::AuthenticationService` struct as its policy value.

See the *CORBA Programmer's Reference, Java* for more details.

By registering an initial reference

You can specify a CSIV2 authentication service object (in C++ and Java) by registering an instance as the `IT_CSIAuthenticationObject` initial reference. This approach is mainly intended for use by Orbix plug-ins.

Default authentication service

If no authentication service is specified, a default implementation is used that always returns `false` in response to `authenticate()` calls.

Orbix Security Framework

In the context of the Orbix Security Framework, the GSP plug-in provides a proprietary implementation of the CSIV2 authentication service that delegates authentication to the Orbix security service.

Sample implementation

A sample implementation of a CSIV2 authentication service can be found in the following demonstration directory:

`ASPIInstallDir/asp/Version/demos/corba/tls/csv2/java/src/csv2`

Providing a Username and Password

Overview

This section explains how a user can provide a username and a password for CSIV2 authentication (logging on) as an application starts up. CSIV2 mandates the use of the GSSUP standard for transmitting a username/password pair between a client and a server.

CSIV2 principal sponsor

The *CSIV2 principal sponsor* is a piece of code embedded in the CSI plug-in that obtains authentication information for an application. It is configured by setting variables in the Orbix configuration. The great advantage of the CSIV2 principal sponsor is that it enables you to provide authentication data for security unaware applications, just by modifying the configuration.

The following configuration file extract shows you how to enable the CSIV2 principal sponsor for GSSUP-style authentication (assuming the application is already configured to load the CSI plug-in):

```
# Orbix configuration file
principal_sponsor:csi:use_principal_sponsor = "true";
principal_sponsor:csi:use_method_id = "GSSUPMech";
```

Credentials sharing

Normally, when you specify an own credential using the CSI principal sponsor, the credential is available only to the ORB that created it. By setting the `plugins:security:share_credentials_across_orbs` variable to `true`, however, the own credentials created by one ORB are automatically made available to any other ORBs that are configured to share credentials.

Logging in

The GSSUP username and password can be provided in one of the following ways:

- [From a dialog prompt.](#)
- [Directly in configuration.](#)
- [By programming.](#)

From a dialog prompt

If the login data are not specified in configuration, the CSiv2 principal sponsor will prompt the user for the username, password, and domain as the application starts up. The dialog prompt is displayed if the client supports the `EstablishTrustInClient CSiv2` association option and one or more of the `principal_sponsor:csi:auth_method_data` fields are missing (username, password, or domain).

C++ Applications

When a C++ application starts up, the user is prompted for the username and password at the command line as follows:

```
Please enter username :  
Enter password :
```

Java Applications

The following dialog window pops up to prompt the user for the username, password, and domain name:

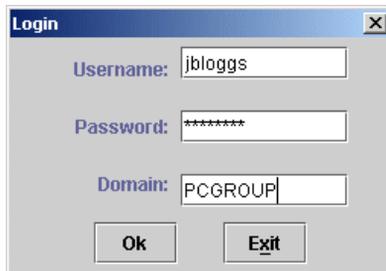


Figure 18: Java Dialog Window for GSSUP Username and Password

Note: The password is not checked until the client communicates with a server secured by CSIV2. Hence, the dialog is unable to provide immediate confirmation of a user's password and a mis-typed password will not be detected until the client begins communicating with the server.

Directly in configuration

The username, password, and domain can be specified directly in the `principal_sponsor:csi:auth_method_data` configuration variable. For example, the CSIV2 principal sponsor can be configured as follows:

```
# Orbix configuration file
principal_sponsor:csi:use_principal_sponsor = "true";
principal_sponsor:csi:use_method_id = "GSSUPMech";
principal_sponsor:csi:auth_method_data = ["username=User",
    "password=Pass", "domain=AuthDomain"];
```

In this example, the `auth_method_data` variable specifies a *User* username, *Pass* password, and *AuthDomain* domain.

WARNING: Storing the password directly in configuration is not recommended for deployed systems. The password is in plain text and could be read by anyone.

By programming

A CORBA application developer can optionally specify the GSSUP username, password and domain name by programming—see [“Creating CSIV2 Credentials” on page 217](#).

In this case, an administrator should ensure that the CSIV2 principal sponsor is disabled for the application. Either the `principal_sponsor:csi:use_principal_sponsor` variable can be set to `false`, or the CSIV2 principal sponsor variables can be removed from the application's configuration.

The best approach is to set the `principal_sponsor:csi:use_principal_sponsor` variable to `false` in the application's configuration scope. For example:

```
# Orbix configuration file
outer_config_scope {
  ...
  my_app_config_scope {
    principal_sponsor:csi:use_principal_sponsor = "false";
    ...
  };
  ...
};
```

This ensures that the principal sponsor cannot be enabled accidentally by picking up configuration variables from the outer configuration scope.

Sample Configuration

Overview

This section provides complete sample configurations, on both the client side and the server side, for the scenario described in [“CSIPv2 Authentication Scenario” on page 158](#).

In this section

This section contains the following subsections:

Sample Client Configuration	page 173
Sample Server Configuration	page 175

Sample Client Configuration

Overview

This section describes a sample client configuration for CSiv2 authentication over transport which has the following features:

- The `iiop_tls` and `csi` plug-ins are loaded into the application.
- The client supports the SSL/TLS `EstablishTrustInTarget` association option.
- The client supports the CSiv2 authentication over transport `EstablishTrustInClient` association option.
- The username and password are specified using the CSiv2 principal sponsor.

Configuration sample

The following sample shows the configuration of a client application that uses CSiv2 authentication over transport to authenticate a user, Paul (using the `csiv2.client.paul` ORB name):

```
# Orbix configuration file
csiv2
{
  orb_plugins = ["local_log_stream", "iiop_profile", "giop",
               "iiop_tls", "csi"];
  event_log:filters = ["IT_CSI=*", "IT_TLS=*", "IT_IIOP_TLS=*",
                     "IT_ATLI_TLS=*"];
  binding:client_binding_list = ["GIOP+EGMIOP",
                                "OTS+POA_Coloc", "POA_Coloc", "OTS+TLS_Coloc+POA_Coloc",
                                "TLS_Coloc+POA_Coloc", "GIOP+SHMIOP", "CSI+OTS+GIOP+IIOP",
                                "CSI+GIOP+IIOP", "CSI+OTS+GIOP+IIOP_TLS",
                                "CSI+GIOP+IIOP_TLS"];
  binding:server_binding_list = ["CSI"];

  client
  {
    policies:iiop_tls:client_secure_invocation_policy:supports
    = ["Integrity", "Confidentiality", "DetectReplay",
      "DetectMisordering", "EstablishTrustInTarget"];
    policies:iiop_tls:client_secure_invocation_policy:requires
    = ["Integrity", "Confidentiality", "DetectReplay",
      "DetectMisordering"];
  }
}
```

```
    paul
    {
        policies:csi:auth_over_transport:client_supports =
["EstablishTrustInClient"];
        policies:csi:auth_over_transport:target_requires =
["EstablishTrustInClient"];

        principal_sponsor:csi:use_principal_sponsor = "true";
        principal_sponsor:csi:auth_method_id = "GSSUPMech";
        principal_sponsor:csi:auth_method_data =
["username=Paul", "password=password", domain="DEFAULT"];
    };
};
```

Sample Server Configuration

Overview

This section describes a sample server configuration for CSIv2 authentication over transport which has the following features:

- The `iiop_tls` and `csi` plug-ins are loaded into the application.
 - The server supports the SSL/TLS `EstablishTrustInTarget` and `EstablishTrustInClient` association options.
 - The server's X.509 certificate is specified using the SSL/TLS principal sponsor.
 - The server supports the CSIv2 authentication over transport `EstablishTrustInClient` association option.
-

Configuration sample

The following sample shows the configuration of a server application that supports CSIv2 authentication over transport (using the `csiv2.server` ORB name):

```
# Orbix configuration file
csiv2
{
  orb_plugins = ["local_log_stream", "iiop_profile", "giop",
               "iiop_tls", "csi"];
  event_log:filters = ["IT_CSI=*", "IT_TLS=*", "IT_IIOP_TLS=*",
                    "IT_ATLI_TLS=*"];
  binding:client_binding_list = ["GIOP+EGMIOP",
                                "OTS+POA_Coloc", "POA_Coloc", "OTS+TLS_Coloc+POA_Coloc",
                                "TLS_Coloc+POA_Coloc", "GIOP+SHMIOP", "CSI+OTS+GIOP+IIOP",
                                "CSI+GIOP+IIOP", "CSI+OTS+GIOP+IIOP_TLS",
                                "CSI+GIOP+IIOP_TLS"];
  binding:server_binding_list = ["CSI"];

  server
  {
    policies:iiop_tls:target_secure_invocation_policy:supports
    = ["Integrity", "Confidentiality", "DetectReplay",
      "DetectMisordering", "EstablishTrustInTarget",
      "EstablishTrustInClient"];
    policies:iiop_tls:target_secure_invocation_policy:requires
    = ["Integrity", "Confidentiality", "DetectReplay",
      "DetectMisordering"];
  }
}
```

```
principal_sponsor:use_principal_sponsor = "true";
principal_sponsor:auth_method_id = "pkcs12_file";
principal_sponsor:auth_method_data =
["filename=C:\ASPInstallDir\asp\6.3\etc\tls\x509\certs\demos\bank_server.p12", "password=bankserverpass"];

policies:csi:auth_over_transport:target_supports =
["EstablishTrustInClient"];
policies:csi:auth_over_transport:authentication_service =
"csiv2.AuthenticationServiceObject";
policies:csi:auth_over_transport:server_domain_name =
"DEFAULT";
};
};
```

Configuring CSiv2 Identity Assertion

This chapter explains the concepts underlying the CSiv2 identity assertion (or delegation) mechanism and provides details of how to configure your applications to use this mechanism.

In this chapter

This chapter discusses the following topics:

CSiv2 Identity Assertion Scenario	page 178
SSL/TLS Prerequisites	page 182
Enabling CSiv2 Identity Assertion	page 184
Sample Configuration	page 186

CSIV2 Identity Assertion Scenario

Overview

This section describes a typical CSIV2 identity assertion scenario, involving a client, an intermediate server, and a target server. Once the client has authenticated itself to the intermediate server, the intermediate server can impersonate the client by including an *identity token* in the requests that it sends to the target server. The intermediate server thus acts as a proxy (or delegate) server.

Identity assertion

The CSIV2 *identity assertion* mechanism provides the basis for a general-purpose delegation or impersonation mechanism. Identity assertion is used in the context of a system where a client invokes an operation on an intermediate server which then invokes an operation on a target server (see [Figure 19](#)). When making a call on the target, the client identity (which is authenticated by the intermediate server) can be forwarded by the intermediate to the target. This enables the intermediate to impersonate the client.

Dependency on SSL/TLS

The CSIV2 identity assertion mechanism relies on SSL/TLS to provide the the following security features at the transport layer (between the intermediate server and the target server):

- Authentication of the target server to the intermediate server.
- Authentication of the intermediate server to the target server.
- Privacy of communication.
- Message integrity.

CSlv2 scenario

Figure 19 shows a typical scenario for CSlv2 identity assertion:

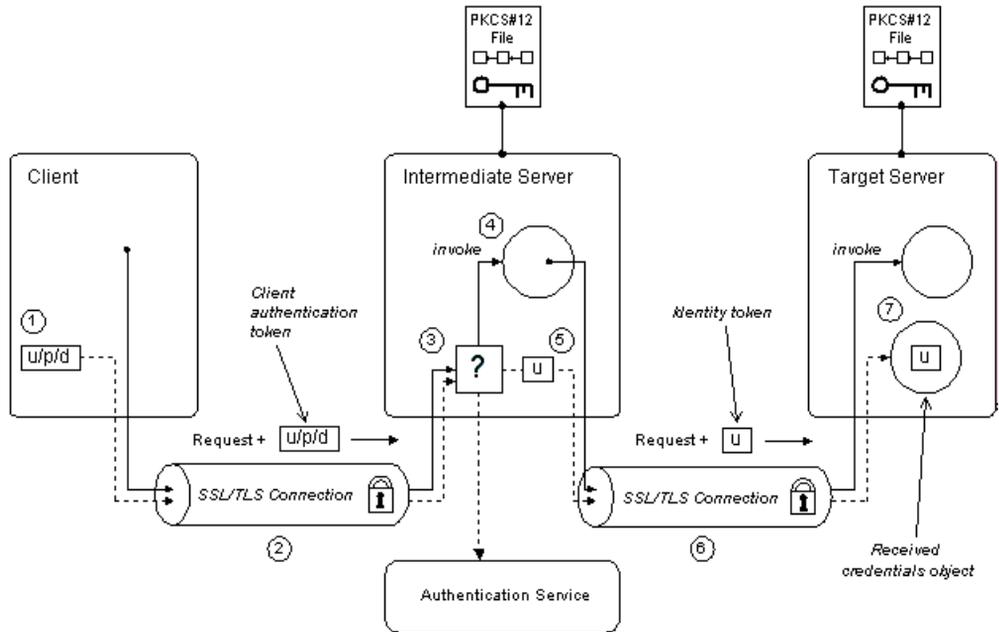


Figure 19: CSlv2 Identity Assertion Scenario

How CSlv2 identity assertion proceeds

As shown in Figure 19 on page 179, the identity assertion mechanism proceeds as follows:

Stage	Description
1	When a client initiates an operation invocation on the intermediate, the client's CSI plug-in inserts a client authentication token (containing username/password/domain) into the GIOP request message.

Stage	Description
2	The request, together with the client authentication token, is sent over the SSL/TLS connection. The SSL/TLS connection provides privacy and message integrity, ensuring that the username and password cannot be read by eavesdroppers.
3	Before permitting the request to reach the target object in the intermediate, the intermediate's CSI plug-in calls the authentication service to check the username/password combination.
4	If the username/password combination are authenticated successfully, the request is allowed to reach the object; otherwise the request is blocked and an error is returned to the client.
5	Within the context of the current invocation, the intermediate server invokes an operation on the target server. Because identity assertion has been enabled on the intermediate server, the intermediate's CSI plug-in extracts the client username from the received GSSUP credentials, creates an <i>identity token</i> containing this username, and then inserts the identity token into the GIOP request message.
6	The request, together with the identity token, is sent over the SSL/TLS connection. The SSL/TLS connection provides privacy message integrity, and mutual authentication between the intermediate and the target.
7	When the request arrives at the target server, the asserted identity is extracted and made available to the target through the CORBA received credentials object—see “Retrieving Received Credentials” on page 230 .

SSL/TLS connection

The intermediate server and target server should both be configured to use a secure SSL/TLS connection. In this scenario, the intermediate-to-target SSL/TLS connection is configured for mutual authentication.

See [“SSL/TLS Prerequisites” on page 182](#) for details of the SSL/TLS configuration for this scenario.

Identity token

An *identity token* can contain one of the following types of identity token:

- `ITTAbsent`—if no identity token is included in the GIOP message sent by the intermediate server (for example, if CSlv2 identity assertion is disabled in the intermediate server).
- `ITTAnonymous`—if the intermediate server is acting on behalf of an anonymous, unauthenticated client.
- `ITTPrincipalName`—if the intermediate server is acting on behalf of an authenticated client. In this case, the client identity contains the following data:
 - ◆ `GSSUP username`—automatically extracted from the GSSUP client authentication token received from the client.
 - ◆ `Subject DN`—if the intermediate server authenticates the client using an X.509 certificate, but not using a username and password, the intermediate would forward on an identity token containing the subject DN from the client certificate.

Received credentials

The *received credentials* is an object, of `SecurityLevel2::ReceivedCredentials` type, defined by the OMG CORBA Security Service that encapsulates the security credentials received from a client. In this scenario, the target server is programmed to access the asserted identity using the received credentials.

For details of how to access the asserted identity through the received credentials object, see [“Retrieving Received Credentials from the Current Object”](#) on page 231.

SSL/TLS Prerequisites

Overview

The CSiv2 identity assertion mechanism relies on SSL/TLS to provide the following security features at the transport layer (between the intermediate server and the target server):

- Authentication of the target server to the intermediate server.
 - Authentication of the intermediate server to the target server.
 - Privacy of communication.
 - Message integrity.
-

SSL/TLS mutual authentication

For the scenario depicted in [Figure 19 on page 179](#), the SSL/TLS connection between the intermediate and the target server is configured for mutual authentication. The SSL/TLS configuration can be summarized as follows:

- *Intermediate server SSL/TLS configuration*—the intermediate server requires confidentiality, message integrity, and the `EstablishTrustInTarget` SSL/TLS association option. An X.509 certificate is provided, which enables the intermediate server to be authenticated both by the client and by the target server.
- *Target server SSL/TLS configuration*—the server requires confidentiality, message integrity, and the `EstablishTrustInClient` SSL/TLS association option. An X.509 certificate is provided, which enables the target server to be authenticated by the intermediate server.

See [“Sample Intermediate Server Configuration” on page 189](#) for a detailed example of the SSL/TLS configuration in this scenario.

See [“SSL/TLS Administration” on page 97](#) for complete details of configuring and administering SSL/TLS.

Setting certificate constraints

In the scenario depicted in [Figure 19 on page 179](#), the target server grants a special type of privilege (backward trust) to the intermediate server—that is, the target accepts identities asserted by the intermediate without getting the chance to authenticate these identities itself. It is, therefore, recommended to set the certificate constraints policy on the target server to restrict the range of applications that can connect to it.

The certificate constraints policy prevents connections being established to the target server, unless the ASN.1 Distinguished Name from the subject line of the incoming X.509 certificate conforms to a certain pattern.

See [“Applying Constraints to Certificates” on page 144](#) for further details.

Principal sponsor configuration

In this scenario, the SSL/TLS principal sponsor needs to be enabled in the intermediate server and in the target server.

See [“Specifying an Application’s Own Certificate” on page 139](#) and for further details.

Note: The SSL/TLS principal sponsor is completely independent of the CSiv2 principal sponsor (see [“Providing a Username and Password” on page 168](#)). It is possible, therefore, to enable both of the principal sponsors within the same application.

Enabling CSiv2 Identity Assertion

Overview

Based on the sample scenario depicted in [Figure 19 on page 179](#), this section describes the basic configuration variables that enable CSiv2 identity assertion. These variables on their own, however, are by no means sufficient to configure a system to use CSiv2 identity assertion. For a complete example of configuring CSiv2 identity assertion, see [“Sample Configuration” on page 186](#).

Loading the CSI plug-in

To enable CSiv2, you must include the `csi` plug-in in the `orb_plugins` list in your Orbix configuration. The `binding:client_binding_list` and `binding:server_binding_list` must also be initialized with the proper list of interceptor combinations.

Sample settings for these configuration variables can be found in the `demos.tls.csiv2` configuration scope of your Orbix configuration. For example, you can load the `csi` plug-in with the following configuration:

```
# Orbix configuration file
csiv2 {
  orb_plugins = ["local_log_stream", "iiop_profile", "giop",
               "iiop_tls", "csi"];

  binding:client_binding_list = ["GIOP+EGMIOP",
                                "OTS+POA_Coloc", "POA_Coloc", "OTS+TLS_Coloc+POA_Coloc",
                                "TLS_Coloc+POA_Coloc", "GIOP+SHMIOP", "CSI+OTS+GIOP+IIOP",
                                "CSI+GIOP+IIOP", "CSI+OTS+GIOP+IIOP_TLS",
                                "CSI+GIOP+IIOP_TLS"];

  binding:server_binding_list = ["CSI"];
  ...
};
```

Intermediate server configuration

The intermediate server can be configured to support CSiv2 identity assertion, as follows:

```
# Orbix configuration file
policies:csi:attribute_service:client_supports =
  ["IdentityAssertion"];
```

Intermediate server CSIV2 association options

Including the `IdentityAssertion` CSIV2 association option in the `policies:csi:attribute_service:client_supports` list indicates that the application supports CSIV2 identity assertion when acting as a client.

Target server configuration

The target server can be configured to support CSIV2 identity assertion, as follows:

```
# Orbix configuration file
policies:csi:attribute_service:target_supports =
  ["IdentityAssertion"];
```

Target server CSIV2 association options

Including the `IdentityAssertion` CSIV2 association option in the `policies:csi:attribute_service:target_supports` list indicates that the application supports CSIV2 identity assertion when acting as a server.

Sample Configuration

Overview

This section provides complete sample configurations, covering the client, the intermediate server, and the target server, for the scenario described in [“CSlv2 Identity Assertion Scenario” on page 178](#).

In this section

This section contains the following subsections:

Sample Client Configuration	page 187
Sample Intermediate Server Configuration	page 189
Sample Target Server Configuration	page 191

Sample Client Configuration

Overview

This section describes a sample client configuration for the CSiv2 identity assertion scenario. In this part of the scenario, the client is configured to use CSiv2 authentication over transport, as follows:

- The `iiop_tls` and `csi` plug-ins are loaded into the application.
- The client supports the SSL/TLS `EstablishTrustInTarget` association option.
- The client supports the CSiv2 authentication over transport `EstablishTrustInClient` association option.
- The username and password are specified using the CSiv2 principal sponsor.

Configuration sample

The following sample shows the configuration of a client application that uses CSiv2 authentication over transport to authenticate a user, Paul (using the `csiv2.client.paul` ORB name):

```
# Orbix configuration file
csiv2
{
  orb_plugins = ["local_log_stream", "iiop_profile", "giop",
               "iiop_tls", "csi"];
  event_log:filters = ["IT_CSI=*", "IT_TLS=*", "IT_IIOP_TLS=*",
                    "IT_ATLI_TLS=*"];
  binding:client_binding_list = ["GIOP+EGMIOP",
                                "OTS+POA_Coloc", "POA_Coloc", "OTS+TLS_Coloc+POA_Coloc",
                                "TLS_Coloc+POA_Coloc", "GIOP+SHMIOP", "CSI+OTS+GIOP+IIOP",
                                "CSI+GIOP+IIOP", "CSI+OTS+GIOP+IIOP_TLS",
                                "CSI+GIOP+IIOP_TLS"];
  binding:server_binding_list = ["CSI"];

  client
  {
    policies:iiop_tls:client_secure_invocation_policy:supports
  = ["Integrity", "Confidentiality", "DetectReplay",
    "DetectMisordering", "EstablishTrustInTarget"];
    policies:iiop_tls:client_secure_invocation_policy:requires
  = ["Integrity", "Confidentiality", "DetectReplay",
    "DetectMisordering"];
  }
}
```

```
    paul
    {
        policies:csi:auth_over_transport:client_supports =
["EstablishTrustInClient"];

        principal_sponsor:csi:use_principal_sponsor = "true";
        principal_sponsor:csi:auth_method_id = "GSSUPMech";
        principal_sponsor:csi:auth_method_data =
["username=Paul", "password=password", "domain=DEFAULT"];
    };
};
```

Sample Intermediate Server Configuration

Overview

This section describes a sample intermediate server configuration for CSiv2 identity assertion which has the following features:

- The `iiop_tls` and `csi` plug-ins are loaded into the application.
 - In the role of server, the intermediate server supports the SSL/TLS `EstablishTrustInTarget` and `EstablishTrustInClient` association options.
 - In the role of client, the intermediate server supports the SSL/TLS `EstablishTrustInTarget` and `EstablishTrustInClient` association options.
 - The intermediate server's X.509 certificate is specified using the SSL/TLS principal sponsor.
 - In the role of server, the intermediate server supports the CSiv2 authentication over transport `EstablishTrustInClient` association option.
 - In the role of client, the intermediate server supports the CSiv2 `IdentityAssertion` association option.
-

Configuration sample

The following sample shows the configuration of an intermediate server application that supports CSiv2 authentication over transport (when acting as a server) and identity assertion (when acting as a client). In this example, the server executable should use the `csiv2.intermed_server` ORB name:

```
# Orbix configuration file
csiv2
{
  orb_plugins = ["local_log_stream", "iiop_profile", "giop",
               "iiop_tls", "csi"];
  event_log:filters = ["IT_CSI=*", "IT_TLS=*", "IT_IIOP_TLS=*",
                     "IT_ATLI_TLS=*"];
  binding:client_binding_list = ["GIOP+EGMIOP",
                                "OTS+POA_Coloc", "POA_Coloc", "OTS+TLS_Coloc+POA_Coloc",
                                "TLS_Coloc+POA_Coloc", "GIOP+SHMIOP", "CSI+OTS+GIOP+IIOP",
                                "CSI+GIOP+IIOP", "CSI+OTS+GIOP+IIOP_TLS",
                                "CSI+GIOP+IIOP_TLS"];
  binding:server_binding_list = ["CSI"];
}
```

```

intermed_server
{
    policies:iiop_tls:target_secure_invocation_policy:supports
= ["Integrity", "Confidentiality", "DetectReplay",
"DetectMisordering", "EstablishTrustInTarget",
"EstablishTrustInClient"];
    policies:iiop_tls:target_secure_invocation_policy:requires
= ["Integrity", "Confidentiality", "DetectReplay",
"DetectMisordering"];
    policies:iiop_tls:client_secure_invocation_policy:supports
= ["Integrity", "Confidentiality", "DetectReplay",
"DetectMisordering", "EstablishTrustInTarget",
"EstablishTrustInClient"];
    policies:iiop_tls:client_secure_invocation_policy:requires
= ["Integrity", "Confidentiality", "DetectReplay",
"DetectMisordering"];

    principal_sponsor:use_principal_sponsor = "true";
    principal_sponsor:auth_method_id = "pkcs12_file";
    principal_sponsor:auth_method_data =
["filename=C:\ASPInstallDir\art\6.3\etc\tls\x509\certs\demos\b
ank_server.p12", "password=bankserverpass"];

    policies:csi:attribute_service:client_supports =
["IdentityAssertion"];
    policies:csi:auth_over_transport:target_supports =
["EstablishTrustInClient"];
    policies:csi:auth_over_transport:target_requires =
["EstablishTrustInClient"];

    policies:csi:auth_over_transport:authentication_service =
"csiv2.AuthenticationServiceObject";
    policies:csi:auth_over_transport:server_domain_name =
"DEFAULT";
};
};

```

Sample Target Server Configuration

Overview

This section describes a sample target server configuration for CSiv2 identity assertion which has the following features:

- The `iiop_tls` and `csi` plug-ins are loaded into the application.
- The server supports the SSL/TLS `EstablishTrustInTarget` and `EstablishTrustInClient` association options.
- The server *requires* the SSL/TLS `EstablishTrustInClient` association option.
- The server's X.509 certificate is specified using the SSL/TLS principal sponsor.
- The intermediate server supports the CSiv2 `IdentityAssertion` association option.

Configuration sample

The following sample shows the configuration of a target server application that supports identity assertion (using the `csiv2.target_server` ORB name).

```
# Orbix configuration file
csiv2
{
  orb_plugins = ["local_log_stream", "iiop_profile", "giop",
               "iiop_tls", "csi"];
  event_log:filters = ["IT_CSI=*", "IT_TLS=*", "IT_IIOP_TLS=*",
                     "IT_ATLI_TLS=*"];
  binding:client_binding_list = ["GIOP+EGMIOP",
                                "OTS+POA_Coloc", "POA_Coloc", "OTS+TLS_Coloc+POA_Coloc",
                                "TLS_Coloc+POA_Coloc", "GIOP+SHMIOP", "CSI+OTS+GIOP+IIOP",
                                "CSI+GIOP+IIOP", "CSI+OTS+GIOP+IIOP_TLS",
                                "CSI+GIOP+IIOP_TLS"];
  binding:server_binding_list = ["CSI"];

  target_server
  {
    policies:iiop_tls:target_secure_invocation_policy:supports
    = ["Integrity", "Confidentiality", "DetectReplay",
      "DetectMisordering", "EstablishTrustInTarget",
      "EstablishTrustInClient"];
  }
}
```

```
    policies:iop_tls:target_secure_invocation_policy:requires
= ["Integrity", "Confidentiality", "DetectReplay",
  "DetectMisordering", "EstablishTrustInClient"];

    principal_sponsor:use_principal_sponsor = "true";
    principal_sponsor:auth_method_id = "pkcs12_file";
    principal_sponsor:auth_method_data =
["filename=C:\ASPInstallDir\art\6.3\etc\tls\x509\certs\demos\b
ank_server.p12", "password=bankserverpass"];
    policies:csi:attribute_service:target_supports =
["IdentityAssertion"];
  };
};
```

Part 5

CORBA Security Programming

In this part

This part contains the following chapters:

Programming Policies	page 195
Authentication	page 209
Validating Certificates	page 233

Programming Policies

You can customize the behavior of secure CORBA applications by setting policies programmatically.

In this chapter

This chapter discusses the following topics:

Setting Policies	page 196
Programmable SSL/TLS Policies	page 199
Programmable CSIV2 Policies	page 206

Setting Policies

Overview

This section provides a brief overview of how to set CORBA policies by programming. An example, in C++ and Java, is provided that shows how to set a CORBA policy at the ORB level.

How to program CORBA policies is described in more detail in the *CORBA Programmer's Guide*.

Client-side policy levels

You can set client-side policies at any of the following levels:

- ORB
 - Thread
 - Object (for client-side proxies).
-

Server-side policy levels

You can set server-side policies at any of the following levels:

- ORB
 - POA
-

Policy management

As described in the *CORBA Programmer's Guide*, you can set a policy at each level using the appropriate policy management object as listed in [Table 8](#).

Table 8: *Policy Management Objects*

Policy Level	Policy Management Object
ORB	<code>CORBA::PolicyManager</code>
Thread	<code>CORBA::PolicyCurrent</code>
POA	<code>PortableServer::POA::create_POA()</code>
Client-side proxy	<code>(ObjectRef)._set_policy_overrides()</code>

C++ Example

The following C++ example shows how to set an SSL/TLS certificate constraints policy at the ORB level:

Example 11: C++ Example of Setting ORB-Level Policies

```
//C++
...
CORBA::Any          any;
CORBA::PolicyList  orb_policies;
orb_policies.length(1);
1  CORBA::Object_var  object =
    global_orb->resolve_initial_references("ORBPolicyManager");
CORBA::PolicyManager_var  policy_mgr =
    CORBA::PolicyManager::_narrow(object);

2  IT_TLS_API::CertConstraints  cert_constraints;
    cert_constraints.length(1);

3  cert_constraints[0] = CORBA::string_dup(
    "C=US,ST=Massachusetts,O=ABigBank*,OU=Administration"
    );

4,5  any <<= cert_constraints;
    orb_policies[0] = global_orb->create_policy(
    IT_TLS_API::TLS_CERT_CONSTRAINTS_POLICY, any
    );

6  policy_mgr->set_policy_overrides(
    orb_policies, CORBA::ADD_OVERRIDE
    );
```

Setting a Policy at ORB Level

The programming steps in the preceding examples, “C++ Example” on [page 197](#), can be explained as follows:

1. Retrieve the ORB policy manager.
2. Create an instance of the policy that you are to adjust, based on the Orbix IDL (see the *CORBA Programmer’s Reference*).
3. Set your new values on this policy.
4. Create an ORB policy object using the `CORBA::ORB::create_policy()` operation and provide your new policy as a parameter.
5. Add the policy to a `PolicyList` object.

6. Use the `PolicyManager::set_policy_overrides()` operation to set the new `PolicyList` on the ORB.

Programmable SSL/TLS Policies

Overview

This section gives a brief overview of the different kinds of programmable SSL/TLS policy and discusses how these policies interact with each other and with policies set in configuration.

For more details of these SSL/TLS policies, consult the relevant sections of the *CORBA Programmer's Reference*.

In this section

This section contains the following subsections:

Introduction to SSL/TLS Policies	page 200
The QOPPolicy	page 202
The EstablishTrustPolicy	page 203
The InvocationCredentialsPolicy	page 204
Interaction between Policies	page 205

Introduction to SSL/TLS Policies

Configuring or programming policies

You can use policies to govern security behavior in Orbix and most of these policies can be set through the Orbix configuration file (see [“policies” on page 272](#)).

However, policies set with the configuration file only apply at the ORB level. If you develop security-aware applications, you can add a finer level of security to objects by programming policies in your application code.

Augmenting minimum levels of security

You can use the CORBA policy IDL and the TLS policy IDL to refine the security features that your objects require. Follow these steps:

1. Consider what are the minimum security levels set for objects in your system.
2. Add to these minimum levels, by adding the available programmable policies to your application code.

Note: Examples of configuring policies programmatically can be found in the TLS policy demo, in the `ASPInstallDir/asp/6.3/demos/tls/policy` directory.

What are the minimum security levels for objects?

You can set the minimum levels of security that objects require with *secure invocation policies*. There are two types of secure invocation policy:

- `Security::SecClientSecureInvocation`
- `Security::SecTargetSecureInvocation`

You can apply values for these in the Orbix configuration file, as discussed in [“Setting Association Options” on page 106](#), or by programming policies.

It is important to remember that by programming policies you can only add more security to the minimum required in the configuration; you cannot reduce the minimum required security by programming.

Required and supported security features

Any object, can have the following dispositions to a security feature:

- If the object *requires* a certain type of security, that requirement must be complied with before a call to the object succeeds.
- If the object *supports* a certain type of security, that security feature can be used, but does not have to be used.

The QOPPolicy

IDL definition

The `SecurityLevel2::QOPPolicy` policy provides a way to override the client and target secure invocation policies. You can apply four levels of protection defined by the enumerated type, `Security::QOP`, defined as follows:

```
//IDL
module Security {
  ...
  enum QOP {
    SecQOPNoProtection,
    SecQOPIntegrity,
    SecQOPConfidentiality,
    SecQOPIntegrityAndConfidentiality
  };
};
```

Purpose

The `SecurityLevel2::QOPPolicy` is used by security aware applications for two purposes:

- Restricting the types of cipher suites available for consideration.
- Overriding the way in which a specific object is contacted.

Restricting cipher suites

The values allowed for QOP policies are not specific enough to identify particular cipher suites (the mechanism policy can be used for this). However the `QOPPolicy` value can render certain cipher suites inapplicable—see [“Constraints Imposed on Cipher Suites” on page 124](#).

If you set a QOP policy to override an existing QOP policy, the applicable list of cipher suites can be extended as a result.

Over-riding how an object is contacted

When you set a QOP policy override for an object, this results in a new object reference that contains the applicable policies. This means that the QOP policy can conveniently be used to create an insecure object reference (where allowed by the administration policies) that you can use for operations where you wish insecure invocations to take place. The original object reference that contains a higher quality of protection can be used for the more sensitive operations.

The EstablishTrustPolicy

Purpose

You can use the `SecurityLevel2::EstablishTrustPolicy` to control whether server or client authentication is to be enforced.

Both a client and target object can *support* this policy, meaning that, for a client, the client is prepared to authenticate its privileges to the target, and the target supports this.

However, you can also set this policy as *required* for a target policy. This means that a client must authenticate its privileges to the target, before the target will accept the connection.

IDL Definition

The `SecurityLevel2::EstablishTrustPolicy` policy contains an attribute, `trust`, of `Security::EstablishTrust` type that specifies whether trust in client and trust in target is enabled. The `Security::EstablishTrust` type is defined as follows:

```
//IDL
module Security {
...
    struct EstablishTrust {
        boolean trust_in_client;
        boolean trust_in_target;
    };
...
};
```

Structure members

This structure contains the following members:

- The `trust_in_client` element stipulates whether the invocation must select credentials and mechanism that allow the client to be authenticated to the target.
- The `trust_in_target` element stipulates whether the invocation must first establish trust in the target.

Note: Normally, all SSL/TLS cipher suites need to authenticate the target.

The InvocationCredentialsPolicy

Purpose

The `SecurityLevel2::InvocationCredentialsPolicy` policy forces a POA to use specific credentials or to use specific credentials on a particular object. When this object is returned by the `get_policy()` operation, it contains the active credentials that will be used for invocations using this target object reference.

Attribute

The `SecurityLevel2::InvocationCredentialsPolicy` policy has a single attribute, `creds`, that returns a list of `Credentials` objects that are used as invocation credentials for invocations through this object reference.

Setting the policy at object level

An `InvocationCredentialsPolicy` object can be passed to the `set_policy_overrides()` operation to specify one or more `Credentials` objects to be used when calling this target object, using the object reference returned by `set_policy_overrides()`.

Interaction between Policies

Upgrading security

To upgrade an insecure Orbix application to be fully secure using the `QOP` and `EstablishTrust` policies, the application must initially be configured to support the `DetectReply` and the `DetectMisordering` association options. This is because it is not possible to specify the `DetectReply` and `DetectMisordering` association options programatically, but these association options are needed for all the SSL/TLS cipher suites. See [“Constraints Imposed on Cipher Suites” on page 124](#).

No downgrading of security

When you specify the client secure invocation policy and the target secure invocation policy, you are providing your application with its *minimum* security requirements. These minimum requirements must be met by any other specified policies and cannot be weakened. This means that the following policies cannot be specified, if their values would conflict with the corresponding `SecureInvocationPolicy` value:

- `QOPPolicy`
 - `MechanismPolicy`
 - `EstablishTrustPolicy`
-

Compatibility with the mechanism policy value

You cannot specify values for the `QOPPolicy`, `SecureInvocationPolicy` (client and target), or `EstablishTrustPolicy`, if the underlying mechanism policy does not support it. For example, you cannot specify that `Confidentiality` is required, if only `NULL` cipher suites are enabled in the `MechanismPolicy`.

Programmable CSlv2 Policies

Overview

This section gives a brief overview of the programmable CSlv2 policies. These programmable policies provide functionality equivalent to the CSlv2 configuration variables.

For complete details of the CSlv2 policies, see the description of the `IT_CSI` module in the *CORBA Programmer's Reference*.

CSlv2 policies

The following CSlv2 policies can be set programmatically:

- [Client-side CSlv2 authentication policy](#).
 - [Server-side CSlv2 authentication policy](#).
 - [Client-side CSlv2 identity assertion policy](#).
 - [Server-side CSlv2 identity assertion policy](#).
-

Client-side CSlv2 authentication policy

You can set the client-side CSlv2 authentication policy to enable an application to send GSSUP username/password credentials over the wire in a GIOP service context. The programmable client-side CSlv2 authentication policy provides functionality equivalent to setting the following configuration variable:

```
policies:csi:auth_over_transport:client_supports
```

To create a client-side CSlv2 authentication policy, use the following IDL data types from the `IT_CSI` module:

- Policy type constant is `IT_CSI::CSI_CLIENT_AS_POLICY`.
 - Policy data is `IT_CSI::AuthenticationService`.
-

Server-side CSlv2 authentication policy

You can set the server-side CSlv2 authentication policy to enable an application to receive and authenticate GSSUP username/password credentials. The programmable server-side CSlv2 authentication policy provides functionality equivalent to setting the following configuration variables:

```
policies:csi:auth_over_transport:target_supports
policies:csi:auth_over_transport:target_requires
policies:csi:auth_over_transport:server_domain_name
policies:csi:auth_over_transport:authentication_service
```

To create a server-side CSlv2 authentication policy, use the following IDL data types from the `IT_CSI` module:

- Policy type constant is `IT_CSI::CSI_SERVER_AS_POLICY`.
 - Policy data is `IT_CSI::AuthenticationService`.
-

Client-side CSlv2 identity assertion policy

You can set the client-side CSlv2 identity assertion policy to enable an application to send a CSlv2 asserted identity over the wire in a GIOP service context. The programmable client-side CSlv2 identity assertion policy provides functionality equivalent to setting the following configuration variable:

```
policies:csi:attribute_service:client_supports
```

To create a client-side CSlv2 identity assertion policy, use the following IDL data types from the `IT_CSI` module:

- Policy type constant is `IT_CSI::CSI_CLIENT_SAS_POLICY`.
 - Policy data is `IT_CSI::AttributeService`.
-

Server-side CSlv2 identity assertion policy

You can set the server-side CSlv2 identity assertion policy to enable an application to receive a CSlv2 asserted identity. The programmable server-side CSlv2 identity assertion policy provides functionality equivalent to setting the following configuration variable:

```
policies:csi:attribute_service:target_supports
```

To create a server-side CSlv2 identity assertion policy, use the following IDL data types from the `IT_CSI` module:

- Policy type constant is `IT_CSI::CSI_SERVER_SAS_POLICY`.
- Policy data is `IT_CSI::AttributeService`.

Authentication

The Orbix Security Framework protects your applications by preventing principals from making calls to the system unless they authenticate themselves.

In this chapter

This chapter discusses the following topics:

Using the Principal Authenticator	page 210
Using a Credentials Object	page 221
Retrieving Own Credentials	page 223
Retrieving Target Credentials	page 226
Retrieving Received Credentials	page 230

Using the Principal Authenticator

Overview

The principal authenticator is an object that associates secure identities with a CORBA application. This section explains how to use the principal authenticator to create various kinds of credentials.

In this section

This section contains the following subsections:

Introduction to the Principal Authenticator	page 211
Creating SSL/TLS Credentials	page 214
Creating CSIV2 Credentials	page 217

Introduction to the Principal Authenticator

Overview

This section describes the role of the principal authenticator object in creating and authenticating an application's own credentials.

Creating own credentials

There are two alternative ways to create an application's own credentials:

- *By configuration*—that is, by setting the principal sponsor configuration variables. See [“Specifying an Application's Own Certificate” on page 139](#).
 - *By programming*—that is, by calling the `SecurityLevel2::PrincipalAuthenticator::authenticate()` operation directly. This alternative is described here.
-

Principal

A *principal* can be any person or code that wants to use your secure system. The principal must be identified, for example by a user name and password, and authenticated. Once authenticated, your system assigns credentials to that principal, that assert the authenticated identity.

Own credentials

An *own credentials* object, of `SecurityLevel2::Credentials` type, represents a secure identity under whose authority the context is executing. When an application invokes an operation on a remote server, it sends one or more of its own credentials to the server in order to identify itself to the server.

Principal authenticator

The *principal authenticator* is a factory object that creates own credentials and associates them with the current ORB instance. By calling the principal authenticator's `authenticate()` operation multiple times, you can associate a list of own credentials objects with the current ORB.

Note: In terms of the CORBA Security Specification, an ORB object is identified with a *security capsule*. The list of own credentials created by a principal authenticator is implicitly associated with the enclosing security capsule.

Credentials sharing

Normally, when you specify an own credential using the principal authenticator, the credential is available only to the ORB that created it. By setting the `plugins:security:share_credentials_across_orbs` variable to `true`, however, the own credentials created by one ORB are automatically made available to any other ORBs that are configured to share credentials.

Creating own credentials

To create own credentials and make them available to your application, follow these steps:

Step	Action
1	Obtain an initial reference to the <code>SecurityLevel2::SecurityManager</code> object.
2	Acquire a <code>SecurityLevel2::PrincipalAuthenticator</code> object from the security manager.
3	Call the <code>PrincipalAuthenticator::authenticate()</code> operation to authenticate the client principal and create a <code>SecurityLevel2::Credentials</code> own credentials object.
4	If more than one type of own credentials object is needed, call the <code>PrincipalAuthenticator::authenticate()</code> operation again with the appropriate arguments.

Types of credentials

Using the `PrincipalAuthenticator`, you can create the following types of credentials:

- [SSL/TLS own credentials](#).
- [CSlv2 own credentials](#).

SSL/TLS own credentials

An SSL/TLS own credentials contains an X.509 certificate chain and is represented by an object of `IT_TLS_API::TLSCredentials` type.

CSlv2 own credentials

The contents of a CSlv2 own credentials depends on the particular mechanism that is used, as follows:

- Username and password—if the CSlv2 authentication over transport mechanism is used.

- Username only—if the CSiv2 identity assertion mechanism is used.

In both cases, the CSiv2 own credentials is represented by an object of `IT_CSI::CSICredentials` type.

Creating SSL/TLS Credentials

Overview

The following authentication methods are supported for SSL/TLS:

- `IT_TLS_API::IT_TLS_AUTH_METH_LABEL`—enables you to specify the name of a label in a key ring (z/OS only).
- `IT_TLS_API::IT_TLS_AUTH_METH_PKCS12_DER`—enables you to specify an X.509 certificate chain in DER-encoded PKCS#12 format. The PKCS#12 data is provided in the form of an `IT_Certificate::DERData` object. Not supported by Schannel.
- `IT_TLS_API::IT_TLS_AUTH_METH_CERT_CHAIN`—enables you to specify the private key and certificate chain directly as `IT_Certificate::DERData` and `IT_Certificate::X509CertChain` objects, respectively. Not supported by Schannel.
- `IT_TLS_API::IT_TLS_AUTH_METH_CERT_CHAIN_FILE`—enables you to specify the path name of a file containing a PEM-encoded X.509 certificate chain. Not supported by Schannel.
- `IT_TLS_API::IT_TLS_AUTH_METH_PKCS11`—enables you to specify the provider, slot number and PIN for a PKCS#11 smart card. Not supported by Schannel.

C++ example

In the following C++ example, a client principal passes its identity to the principal authenticator in the form of a PKCS#12 file:

Example 12: C++ Example of SSL/TLS Authentication

```
//C++
int pkcs12_login(
    CORBA::ORB_ptr orb,
    const char *pkcs12_filename,
    const char *password
)
{
    CORBA::Any    auth_data;
    CORBA::Any*  continuation_data_ign;
    CORBA::Any*  auth_specific_data_ign;
    Security::AttributeList    privileges; // Empty
1 SecurityLevel2::Credentials_var creds;
    Security::AuthenticationStatus    status;
```

Example 12: C++ Example of SSL/TLS Authentication

```

IT_TLS_API::PKCS12FileAuthData  p12_auth_data;
CORBA::Object_var              obj;
SecurityLevel2::SecurityManager_var security_manager_obj;
SecurityLevel2::PrincipalAuthenticator_var
    principal_authenticator_obj;

2  obj = orb->resolve_initial_references("SecurityManager");
    security_manager_obj = SecurityLevel2::SecurityManager::
        _narrow(obj);

3  principal_authenticator_obj =
    security_manager_obj->principal_authenticator();

p12_auth_data.filename =
    CORBA::string_dup(pkcs12_filename);
p12_auth_data.password =
    CORBA::string_dup(password);
auth_data <<= p12_auth_data;

4  status = principal_authenticator_obj->authenticate(
    IT_TLS_API::IT_TLS_AUTH_METH_PKCS12_FILE,
    "", // The mechanism name.
    NULL, // SecurityName (not used for this method).
    auth_data, // The authentication data for this method of
                // authentication.
    privileges, // Empty list, no privileges are supported
                // by SSL.
    creds,
    continuation_data_ign, // These last two paramaters are
    auth_specific_data_ign // not used by this
                            // mechanism/method combination.
    );
...

```

C++ notes

The preceding C++ example can be explained as follows:

1. Declare an empty credentials object reference to hold the security attributes of this client if login is successful.
2. Obtain an initial reference to the `SecurityManager` object.
3. Acquire a `PrincipalAuthenticator` object from the security manager.

4. Use the `PrincipleAuthenticator` to authenticate the client principal. If this operation returns a value of `Security::SecAuthSuccess`, the security attributes of the authenticated object are stored in the credentials object, `creds`.

Creating CSiv2 Credentials

Overview

The following authentication method is supported for CSiv2:

- `IT_CSI::IT_CSI_AUTH_METH_USERNAME_PASSWORD`—enables you to specify a GSSUP username, password, and domain. The GSSUP authentication data is provided in the form of an `IT_CSI::GSSUPAuthData` object.

C++ example

[Example 13](#) shows how to create CSiv2 credentials in C++, by supplying a username, `<user_name>`, password, `<password>`, and authentication domain, `<domain>`, to the principal authenticator's `authenticate()` operation.

Example 13: C++ Example of CSiv2 Authentication

```
// C++
int
set_csiv2_credential(CORBA::ORB_var orb)
{
    IT_CSI::GSSUPAuthData          csi_gssup_auth_data;
    CORBA::Any                     auth_data;
    CORBA::Any*                    continuation_data_ign;
    CORBA::Any*                    auth_specific_data_ign;
    Security::AttributeList        privileges;
    SecurityLevel2::Credentials_var creds;
    CORBA::String_var              username;
    Security::AuthenticationStatus  status;
    SecurityLevel2::PrincipalAuthenticator_var authenticator;

    try {
        // Get initial reference of SecurityManager
        SecurityLevel2::SecurityManager_var security_manager_obj;

        try
        {
            CORBA::Object_var obj;
            obj = orb->resolve_initial_references(
                "SecurityManager"
            );
            security_manager_obj =
                SecurityLevel2::SecurityManager::_narrow(obj);

```

1

Example 13: C++ Example of CSIV2 Authentication

```

        if (CORBA::is_nil(security_manager_obj))
        {
            cerr << "Unexpected Error. Failed to initialize "
                 "SecurityManager initial reference." << endl;
        }

2       authenticator =
           security_manager_obj->principal_authenticator();
       if (CORBA::is_nil(authenticator))
       {
           // Log error message (not shown) ...
           return -1;
       }
    }
    catch (const CORBA::ORB::InvalidName&)
    {
        // Log error message (not shown) ...
        return -1;
    }

3       username = CORBA::string_dup("<user_name>");
       csi_gssup_auth_data.password =
           CORBA::string_dup("<password>");
       csi_gssup_auth_data.domain =
4       CORBA::string_dup("<domain>");
       auth_data <<= csi_gssup_auth_data;
       ...
5       status = authenticator->authenticate(
           IT_CSI::IT_CSI_AUTH_METH_USERNAME_PASSWORD,
           "", // NOT USED
           username, // GSSUP user name
           auth_data, // GSSUP auth data in an any
           privileges, // NOT USED
           creds, // returned credentials
           continuation_data_ign, // NOT USED
           auth_specific_data_ign // NOT USED
       );

       if (status != Security::SecAuthSuccess)
       {
           // Log error message (not shown) ...
           return -1;
       }
    }
}

```

Example 13: C++ Example of CSrv2 Authentication

```

        if (CORBA::is_nil(security_manager_obj))
        {
            cerr << "Unexpected Error. Failed to initialize "
                 "SecurityManager initial reference." << endl;
        }

2       authenticator =
           security_manager_obj->principal_authenticator();
       if (CORBA::is_nil(authenticator))
       {
           // Log error message (not shown) ...
           return -1;
       }
    }
    catch (const CORBA::ORB::InvalidName&)
    {
        // Log error message (not shown) ...
        return -1;
    }

3       username = CORBA::string_dup("<user_name>");
       csi_gssup_auth_data.password =
           CORBA::string_dup("<password>");
       csi_gssup_auth_data.domain =
4       CORBA::string_dup("<domain>");
       auth_data <<= csi_gssup_auth_data;
       ...
5       status = authenticator->authenticate(
           IT_CSI::IT_CSI_AUTH_METH_USERNAME_PASSWORD,
           "", // NOT USED
           username, // GSSUP user name
           auth_data, // GSSUP auth data in an any
           privileges, // NOT USED
           creds, // returned credentials
           continuation_data_ign, // NOT USED
           auth_specific_data_ign // NOT USED
       );

       if (status != Security::SecAuthSuccess)
       {
           // Log error message (not shown) ...
           return -1;
       }
    }
}

```

Example 13: C++ Example of CSv2 Authentication

```
catch(const CORBA::Exception& ex)
{
    cerr << "Could not set csi credentials, " << ex << endl;
    return -1;
}
return 0;
}
```

C++ notes

The preceding C++ example can be explained as follows:

1. Obtain an initial reference to the `SecurityManager` object.
2. Acquire a `PrincipleAuthenticator` object from the security manager.
3. Create a `GSSUPAuthData` struct containing the GSSUP password, `<password>`, and domain, `<domain>`.
4. Insert the `GSSUPAuthData` struct, `auth_data`, into the `any`, `auth_data_any`.
5. Call `authenticate()` on the `PrincipleAuthenticator` object to authenticate the client principal. If the `authenticate()` operation returns a value of `Security::SecAuthSuccess`, the security attributes of the authenticated object are stored in `creds`.

Using a Credentials Object

What is a credentials object?

A `SecurityLevel2::Credentials` object is a locality-constrained object that represents a particular principal's credential information, specific to the execution context. A `Credentials` object stores security attributes, including authenticated (or unauthenticated) identities, and provides operations to obtain and set the security attributes of the principal it represents.

Credentials types

There are three types of credentials:

- *Own credentials*—identifies the principal under whose authority the context is executing. An own credential is represented by an object of `SecurityLevel2::Credentials` type.
- *Target credentials*—identifies a remote target object. A target credential is represented by an object of `SecurityLevel2::TargetCredentials` type.
- *Received credentials*—identifies the principal that last sent a message to the current execution context (for example, the principal that called a currently executing operation). A received credential is represented by an object of `SecurityLevel2::ReceivedCredentials` type.

How credentials are obtained

Credentials objects are created or obtained as the result of:

- Authentication.
- Asking for a `Credentials` object from a `SecurityLevel2::Current` object or from a `SecurityLevel2::SecurityManager` object.

Accessing the credentials attributes

The security attributes associated with a `Credentials` object can be obtained by calling the `SecurityLevel2::Credentials::get_attributes()` operation, which returns a list of security attributes (of `Security::AttributeList` type).

Standard credentials attributes

Two security attribute types are supported by Orbix (of `Security::SecurityAttributeType` type), as follows:

- `Security::_Public`—present in every `Credentials` object. The value of this attribute is always empty.

Note: The `_` (underscore) prefix in `_Public` is needed to avoid a clash with the IDL keyword, `public`. The underscore prefix is, however, omitted from the corresponding C++ and Java identifiers.

- `Security::AccessId`—present only if the `Credentials` object represents a valid credential (containing an X.509 certificate chain). In SSL/TLS, the value of this attribute is the string form of the subject DN of the first certificate in the certificate chain.

Orbix-specific credentials attributes

Orbix also enables you to access the X.509 certificate chain associated with a `Credentials` object by narrowing the `Credentials` object to one of the following interface types: `IT_TLS_API::Credentials`, `IT_TLS_API::ReceivedCredentials`, or `IT_TLS_API::TargetCredentials`.

Retrieval method summary

The different credentials types can be retrieved in the following ways:

- *Retrieving own credentials*—a client's own credentials can be retrieved from the `SecurityLevel2::SecurityManager` object.
- *Retrieving target credentials*—a client can retrieve target credentials (if they are available) by passing the target's object reference to the `SecurityLevel2::SecurityManager::get_target_credentials()` operation.
- *Retrieving received credentials*—a server can retrieve an authenticated client's credentials from the `SecurityLevel2::Current` object.

Retrieving Own Credentials

Overview

This section describes how to retrieve own credentials from the security manager object and how to access the information contained in the own credentials.

In this section

This section contains the following subsections:

Retrieving Own Credentials from the Security Manager	page 224
Parsing SSL/TLS Own Credentials	page 225

Retrieving Own Credentials from the Security Manager

Overview

This section describes how to retrieve an application's list of own credentials from the security manager object.

The security manager object

The `SecurityLevel2::SecurityManager` object provides access to ORB-specific security information. The attributes and operations of the `SecurityManager` object apply to the current security capsule (that is, ORB or group of credentials-sharing ORBs) regardless of the thread of execution.

Security manager operations and attributes

The attributes and operations on the `SecurityLevel2::SecurityManager` object are described in the *CORBA Programmer's Reference*.

C++ example

In C++, you can retrieve an application's own credentials list as shown in [Example 14](#).

Example 14: Retrieving a C++ Application's Own Credentials List

```
// C++
...
1 CORBA::Object_var obj =
    my_orb->resolve_initial_references("SecurityManager");
  SecurityLevel2::SecurityManager_var security_manager_obj =
    SecurityLevel2::SecurityManager::_narrow(obj);
  if (CORBA::is_nil(security_manager_obj))
  {
    // Error! Deal with failed narrow...
  }
2 SecurityLevel2::CredentialsList_var creds_list =
    security_manager_obj->own_credentials();
...

```

The preceding code example can be described, as follows:

1. The standard string, `SecurityManager`, is used to obtain an initial reference to the `SecurityLevel2::SecurityManager` object.
2. The list of own credentials is obtained from the `own_credentials` attribute of the security manager object.

Parsing SSL/TLS Own Credentials

Overview

This subsection explains how to access the information stored in an SSL/TLS credentials object. If a credentials object obtained from the security manager is of SSL/TLS type, you can narrow the credentials to the `IT_TLS_API::TLSCredentials` type to gain access to its X.509 certificate chain.

C++ example

In C++, if the own credentials list contains a list of SSL/TLS credentials, you can access the credentials as follows:

```
// C++
for (CORBA::ULong i=0; i < creds_list->length(); i++)
{
    // Access the i'th own credentials in the list
    IT_TLS_API::TLSCredentials_var tls_creds =
        IT_TLS_API::TLSCredentials::_narrow(creds_list[i]);
    if (CORBA::is_nil(tls_creds))
    {
        // Error! Deal with failed narrow...
    }

    // Get the first X.509 certificate in the chain
    IT_Certificate::X509Cert_var cert =
        tls_creds->get_x509_cert();

    // Examine the X.509 certificate, etc.
    ...
}
```

Retrieving Target Credentials

Overview

This section describes how to retrieve the target credentials from a particular target object and how to access the information contained in the target credentials.

In this section

This section contains the following subsections:

Retrieving Target Credentials from an Object Reference	page 227
Parsing SSL/TLS Target Credentials	page 229

Retrieving Target Credentials from an Object Reference

Availability of target credentials

Target credentials are available on the client side only if the client is configured to authenticate the remote target object. For almost all SSL/TLS cipher suites and for all SSL/TLS cipher suites currently supported by Orbix ASP this is the case.

When target credentials are available to the client, they are implicitly associated with an object reference.

The TargetCredentials interface

The `SecurityLevel2::TargetCredentials` interface is the standard type used to represent a target credentials object. It is described in the *CORBA Programmer's Reference*.

Interaction with rebind policy

If you are going to retrieve target credentials, you should be aware of the possible interactions with the rebind policy.

WARNING: If you want to check the target credentials, you should ensure that transparent rebinding is disabled by setting the `policies:rebind_policy` configuration variable to `NO_REBIND`. Otherwise, a secure association could close (for example, if automatic connection management is enabled) and rebind to a different server without the client being aware of this.

C++ example

In C++, you can retrieve the target credentials associated with a particular object reference, `target_ref`, as shown in [Example 15](#).

Example 15: C++ *Obtaining Target Credentials*

```
// C++
...
// Given the following prerequisites:
// my_orb - a reference to an ORB instance.
// target_ref - an object reference to a remote, secured object.

CORBA::Object_var obj =
    my_orb->resolve_initial_references("SecurityManager");
SecurityLevel2::SecurityManager_var security_manager_obj =
    SecurityLevel2::SecurityManager::_narrow(obj);
if (CORBA::is_nil(security_manager_obj))
{
    // Error! Deal with failed narrow...
}

SecurityLevel2::TargetCredentials_var target_creds =
    security_manager_obj->get_target_credentials(target_ref);
...
```

Parsing SSL/TLS Target Credentials

Overview

If you want to access the added value Orbix functionality for SSL/TLS target credentials, perform this additional step after obtaining the target credentials (otherwise, you can use the standard `SecurityLevel2::Credentials` interface).

Narrow the `SecurityLevel2::TargetCredentials` object to the `IT_TLS_API::TLSTargetCredentials` type to gain access to its X.509 certificate.

C++ example

In C++, after obtaining a target credentials object, `target_creds`, as shown in [Example 15 on page 228](#), you can access the SSL/TLS specific data as follows:

```
// C++
...
IT_TLS_API::TLSTargetCredentials_var tls_target_creds =
    IT_TLS_API::TLSTargetCredentials::_narrow(target_creds);
if (CORBA::is_nil(tls_target_creds))
{
    // Error! Deal with failed narrow...
}

// Get the first X.509 certificate in the chain
IT_Certificate::X509Cert_var cert =
    tls_target_creds->get_x509_cert();

// Examine the X.509 certificate, etc.
...
```

Retrieving Received Credentials

Overview

This section describes how to retrieve received credentials from the current object and how to access the information contained in the received credentials.

In this section

This section contains the following subsections:

Retrieving Received Credentials from the Current Object	page 231
Parsing SSL/TLS Received Credentials	page 232

Retrieving Received Credentials from the Current Object

Role of the SecurityLevel2::Current object

A security-aware server application can obtain information about the attributes of the calling principal through the `SecurityLevel2::Current` object. The `SecurityLevel2::Current` object contains information about the execution context.

The SecurityLevel2::Current interface

The `SecurityLevel2::Current` interface is described in detail in the *CORBA Programmer's Reference*.

C++ example

In C++, to obtain received credentials, perform the steps shown in [Example 16](#).

Example 16: C++ Retrieving Received Credentials

```
// C++
...
// In the context of an operation/attribute implementation

CORBA::Object_var obj =
    my_orb->resolve_initial_references("SecurityCurrent");
SecurityLevel2::Current_var current_obj =
    SecurityLevel2::Current::_narrow(obj);
if (CORBA::is_nil(current_obj))
{
    // Error! Deal with failed narrow...
}

SecurityLevel2::ReceivedCredentials_var recvd_creds =
    current_obj->received_credentials();
...
```

Parsing SSL/TLS Received Credentials

Overview

If you want to access the added value Orbix functionality for SSL/TLS received credentials, perform this additional step (otherwise, you can use the standard `SecurityLevel2::Credentials` interface).

Narrow the `SecurityLevel2::ReceivedCredentials` object to the `IT_TLS_API::TLSReceivedCredentials` type to gain access to its X.509 certificate (this step is specific to Orbix).

C++ example

In C++, after obtaining a received credentials object, `recvd_creds`, (see [Example 16 on page 231](#)) you can access the SSL/TLS specific data as follows:

```
// C++
...
IT_TLS_API::TLSReceivedCredentials_var tls_recvd_creds =
    IT_TLS_API::TLSReceivedCredentials::_narrow(recvd_creds);
if (CORBA::is_nil(tls_recvd_creds))
{
    // Error! Deal with failed narrow...
}

// Get the first X.509 certificate in the chain
IT_Certificate::X509Cert_var cert =
    tls_recvd_creds->get_x509_cert();

// Examine the X.509 certificate, etc.
...
```

Validating Certificates

During secure authentication, Orbix TLS checks the validity of an application's certificate. This chapter describes how Orbix validates a certificate and how you can use the Orbix API to introduce additional validation to your applications.

In this chapter

This chapter discusses the following topics:

Overview of Certificate Validation	page 234
The Contents of an X.509 Certificate	page 237
Parsing an X.509 Certificate	page 238
Controlling Certificate Validation	page 239
Obtaining an X.509 Certificate	page 247

Overview of Certificate Validation

Certificate validation

The Orbix API allows you to define a certificate validation policy that implements custom validation of certificates. During authentication, Orbix validates a certificate and then passes it to a certificate validation object, if you have specified a certificate validation policy. This functionality is useful in systems that have application-specific requirements for the contents of each certificate.

Validation process

A server sends its certificate to a client during a TLS handshake, as follows:

1. The server obtains its certificate (for example, by reading it from a local file) and transmits it as part of the handshake.
2. The client reads the certificate from the network, checks the validity of its contents, and either accepts or rejects the certificate.

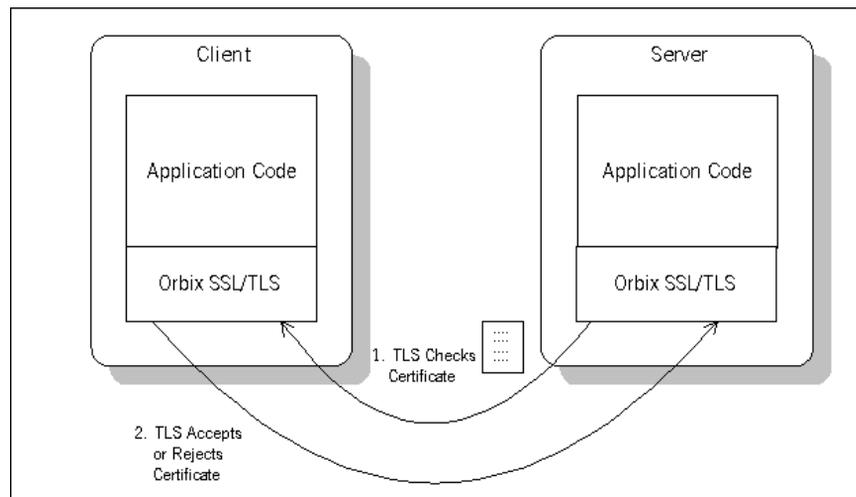


Figure 20: Validating a Certificate

Default validation

The default certificate validation in Orbix checks the following:

- The certificate is a validly constructed X.509 certificate.
- The signature is correct for the certificate.
- The certificate has not expired and is currently valid.
- The certificate chain is validly constructed, consisting of the peer certificate plus valid issuer certificates up to the maximum allowed chain depth.
- If the `CertConstraintsPolicy` has been set, the DN of the received peer certificate is checked to see if it passes *any* of the constraints in the policy conditions. This applies only to the application certificate, not the CA certificates in the chain.

Custom validation

For some applications, it is necessary to introduce additional validation. For example, your client programs might check that each server uses a specific, expected certificate (that is, the distinguished name matches an expected value). Using Orbix, you can perform custom validation on certificates by registering an `IT_TLS_API::CertValidatorPolicy` and implementing an associated `IT_TLS::CertValidator` object.

Example of custom validation

For example, [Figure 21](#) shows the steps followed by Orbix to validate a certificate when a `CertValidatorPolicy` has been registered on the client side:

1. The standard validation checks are applied by Orbix.
2. The certificate is then passed to an `IT_TLS::CertValidator` callback object that performs user-specified validation on the certificate.
3. The user-specified `CertValidator` callback object can decide whether to accept or reject the certificate.
4. Orbix accepts or rejects the certificate.

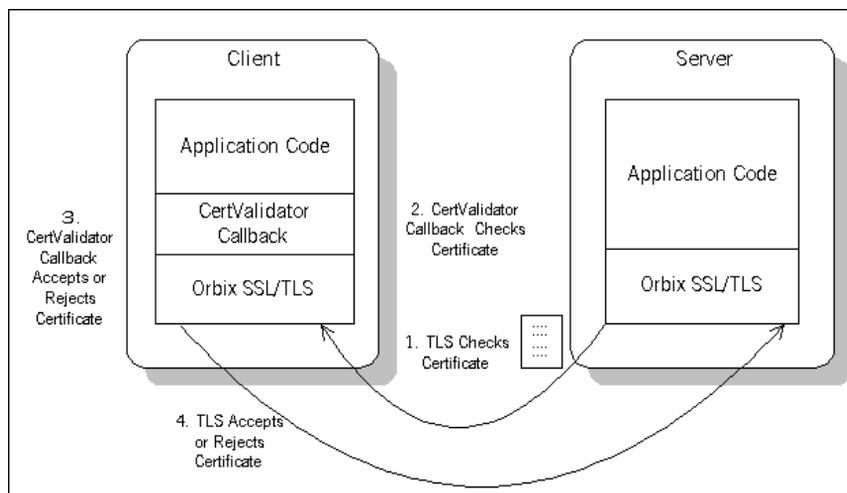


Figure 21: *Using a CertValidator Callback*

The Contents of an X.509 Certificate

Purpose of a certificate

An X.509 certificate contains information about the certificate subject and the certificate issuer (the CA that issued the certificate).

Certificate syntax

A certificate is encoded in Abstract Syntax Notation One (ASN.1), a standard syntax for describing messages that can be sent or received on a network.

Certificate contents

The role of a certificate is to associate an identity with a public key value. In more detail, a certificate includes:

- X.509 version information.
- A *serial number* that uniquely identifies the certificate.
- A *common name* that identifies the subject.
- The *public key* associated with the common name.
- The name of the user who created the certificate, which is known as the *subject name*.
- Information about the *certificate issuer*.
- The signature of the issuer.
- Information about the algorithm used to sign the certificate.
- Some optional X.509 v3 extensions. For example, an extension exists that distinguishes between CA certificates and end-entity certificates.

Parsing an X.509 Certificate

C++ parsing

Orbix ASP provides a high-level set of C++ classes that provide the ability to parse X.509 v3 certificates, including X.509 v3 extensions. When writing your certificate validation functions, you use these classes to examine the certificate contents.

The C++ parsing classes are mapped from the interfaces appearing in the `IT_Certificate` IDL module—see the *CORBA Programmer's Reference*.

Working with distinguished names in C++

An X.509 certificate uses ASN.1 *distinguished name* structures to store information about the certificate issuer and subject. A distinguished name consists of a series of attribute value assertions (AVAs). Each AVA associates a value with a field from the distinguished name.

For example, the distinguished name for a certificate issuer could be represented in string format as follows:

```
/C=IE/ST=Co. Dublin/L=Dublin/O=Orbix Services/OU=PD/CN=Orbix
  Services
```

In this example, AVAs are separated by the `/` character. The first field in the distinguished name is `C`, representing the country of the issuer, and the corresponding value is the country code `IE`. This example distinguished name contains six AVAs.

Extracting distinguished names from certificates in C++

Once you have acquired a certificate, the `IT_Certificate::Certificate` interface permits you to retrieve distinguished names using the `get_issuer_dn_string()` and `get_subject_dn_string()` operations. These operations return an object derived from the `IT_Certificate::AVAList` interface. The `AVAList` interface gives you access to the AVA objects contained in the distinguished name. For more information on these interfaces, see the *CORBA Programmer's Reference*.

Working with X.509 extensions in C++

Some X.509 v3 certificates include extensions. These extensions can contain several different types of information. You can use the `IT_Certificate::ExtensionList` and `IT_Certificate::Extension` interfaces described in the *CORBA Programmer's Reference* to retrieve this information.

Controlling Certificate Validation

Policies used for certificate validation

You can control how your applications handle certificate validation using the following Orbix policies:

<code>CertConstraintsPolicy</code>	Use this policy to apply conditions that peer X.509 certificates must meet to be accepted.
<code>CertificateValidatorPolicy</code>	Use this policy to create customized validations of peer certificate chains.

In this section

This section contains the following subsections:

Certificate Constraints Policy	page 240
Certificate Validation Policy	page 243

Certificate Constraints Policy

Constraints applied to distinguished names

You can impose rules about which peer certificates to accept using certificate constraints. These are conditions imposed on a received certificate subject's distinguished name (DN). Distinguished names are made up of a number of distinct fields, the most common being Organization Unit (OU) and Common Name (CN). Constraints are not applied to all certificates in a received certificate chain, but only to the first in the list, the peer application certificate.

Alternatives ways to set the constraints policy

Use the certificate constraints policy to apply these conditions. You can set this policy in two ways:

- By configuration* This allows you to set constraints at the granularity of an ORB. The same constraints are applied to both client and server peer certificates.
 - By programming* This allows you to set constraints by ORB, thread, POA, or object reference. You can also differentiate between client and server certificates when specifying constraints.
-

Setting the CertConstraintsPolicy by configuration

You can set the `CertConstraintsPolicy` in the configuration file. For example:

```
"C=US,ST=Massachusetts,O=ABigBank*,OU=Administration"
```

In this case, the same constraints string applies to all POAs. If you need different constraints for different POAs then you must supply the policy at POA creation time. For more details, see [“Applying Constraints to Certificates” on page 144](#).

Setting the CertConstraintsPolicy by programming

When you specify a `CertConstraintsPolicy` object on an ORB programmatically, objects created by that ORB apply the certificate constraints to all applications that connect to it.

In the following example, the certificate constraints string specified only allows clients from the Administration Organization unit to connect. The administration user is the only client that has a certificate that satisfies this constraint.

Note: This certificate constraints policy is only relevant if the target object supports client authentication.

C++ example

The following C++ example shows how to set the `CertConstraintsPolicy` programmatically:

Example 17: C++ Example of Setting the `CertConstraintsPolicy`

```
//C++
...
CORBA::Any any;
1 CORBA::PolicyList orb_policies;
  orb_policies.length(1);
2 CORBA::Object_var object =
  global_orb->resolve_initial_references("ORBPolicyManager");
CORBA::PolicyManager_var policy_mgr = CORBA::PolicyManager::
  _narrow(object);
3 IT_TLS_API::CertConstraints cert_constraints;
  cert_constraints.length(1);
  cert_constraints[0] =
  CORBA::string_dup("C=US,ST=Massachusetts,
    O=ABigBank*,OU=Administration");
  any <<= cert_constraints;
4 orb_policies[0] = global_orb->create_policy(IT_TLS_API::
  TLS_CERT_CONSTRAINTS_POLICY, any);
5 policy_mgr->set_policy_overrides(orb_policies, CORBA::
  ADD_OVERRIDE);
```

C++ example description

The preceding C++ example can be explained as follows:

1. Create a `PolicyList` object.
2. Retrieve the `PolicyManager` object.
3. Instantiate a `CertConstraints` data instance (string array).

4. Create a policy using the `CORBA::ORB::create_policy()` operation. The first parameter to this operation sets the policy type to `TLS_CERT_CONSTRAINTS_POLICY`, and the second is an `Any` containing the custom policy.
5. Use the `PolicyManager` to add the new policy override to the Orb scope

Certificate Validation Policy

Certificate validation

Your applications can perform customized validation of peer certificate chains. This enables them, for example, to perform special validation on x.509 v3 extensions or do automatic database lookups to validate subject DNs.

Restrictions on custom certificate validation

The customized certificate validation policy cannot make Orbix accept a certificate that the system has already decided is invalid. It can only reject a certificate that would otherwise have been accepted.

Customizing your applications

To customize your applications, perform the following steps:

Step	Action
1	Derive a class from the CertValidator signature class.
2	Override the validate_cert_chain() operation.
3	Specify the CertValidatorPolicy on the ORB.

Your customized policy is used in addition to the default CertValidatorPolicy.

Derive a class from the CertValidator signature class

In the following example, an implementation class is derived from the `IT_TLS::CertValidator` interface:

```
//C++
class CustomCertValidatorImpl :
    public virtual IT_TLS::CertValidator,
    public virtual CORBA::LocalObject
{
public:

    CORBA::Boolean
    validate_cert_chain(
        CORBA::Boolean chain_is_valid,
        const IT_Certificate::X509CertChain& cert_chain,
```

```

        const IT_TLS::CertChainErrorInfo& error_info
    );
};

```

The class contains your custom version of the `validate_cert_chain()` function.

Override the `validate_cert_chain()` operation

The following is an example custom validation function simply retrieves a name from a certificate:

Example 18: C++ Example of Overriding `validate_cert_chain()`

```

//C++
CORBA::Boolean
CustomCertValidatorImpl::validate_cert_chain(
    CORBA::Boolean chain_is_valid,
    const IT_Certificate::X509CertChain& cert_chain,
    const IT_TLS::CertChainErrorInfo& error_info
)
{
    if (chain_is_valid)
    {
        CORBA::String_var CN;
        1 IT_Certificate::X509Cert_var cert = cert_chain[0];
        2 IT_Certificate::AVAList_var subject =
            cert->get_subject_avalist();

        IT_Certificate::Bytes* subject_string_name;
        3 subject_string_name = subject->convert(IT_Certificate::
            IT_FMT_STRING);

        int len = subject_string_name->length();
        char *str_name = new char[len];
        for (int i = 0; i < len; i++){
            str_name[i] = (char)((*subject_string_name)[i]);
        }
        return chain_is_valid;
    }
}

```

The preceding C++ example can be explained as follows:

1. The certificate is retrieved from the certificate chain.
2. An AVAList (see [“Working with distinguished names in C++” on page 238](#)) containing the distinguished name is retrieved from the certificate.
3. The distinguished name is converted to string format.

Specify the CertValidatorPolicy on the ORB

Once you have devised your custom validation class, create an instance of it and apply it as a policy to the Orb with the policy manager, as shown in the following example:

Example 19: C++ Example of Setting the CertValidatorPolicy

```
//C++
int main(int argc, char* argv[])
{
    CORBA::PolicyTypeSeq types;
    CORBA::PolicyList policies(1);
    CORBA::Any policy_any;
    CORBA::Object_var object;
    CORBA::PolicyManager_var policy_mgr;
    IT_TLS::CertValidator_ptr custom_cert_val_obj;

1   policies.length(1);
   types.length(1);
2   types[0] = IT_TLS_API::TLS_CERT_VALIDATOR_POLICY;

    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

    object = orb->resolve_initial_references("ORBPolicyManager");
3   policy_mgr = CORBA::PolicyManager::_narrow(object);

    // set cert validator policy at ORB scope
4   custom_cert_val_obj = new CustomCertValidatorImpl;
   policy_any <<= custom_cert_val_obj;
5   policies[0] =
   orb->create_policy(IT_TLS_API::TLS_CERT_VALIDATOR_POLICY,
   policy_any);

6   policy_mgr->set_policy_overrides(
       policies,
       CORBA::ADD_OVERRIDE
   );
}
```

Example 19: C++ Example of Setting the *CertValidatorPolicy*

```
...  
}
```

As can be seen from the above example, you can apply the new `CertValidator` policy to the Orb in the same manner as any other Orbix policy:

1. Create a `CORBA::PolicyList` object.
2. Set the type of the appropriate policy slot in the `PolicyList` to `TLS_CERT_VALIDATOR_POLICY`. In this example, the first slot is chosen.
3. Retrieve the `CORBA::PolicyManager` object.
4. Instantiate the custom `IT_TLS::CertValidator` policy object.
5. Create a policy using the `CORBA::ORB::create_policy()` operation. The first parameter to this operation sets the policy type to `TLS_CERT_VALIDATOR_POLICY`, and the second is a `CORBA::Any` containing the custom policy.
6. Use the `PolicyManager` to add the new policy override to the ORB scope.

Obtaining an X.509 Certificate

Alternative ways of obtaining certificates

You can obtain a certificate in the following ways:

- Using the `IT_TLS_API::TLSCredentials` interface, which enables you to retrieve X.509 certificates from a credentials object—see [“Retrieving Own Credentials” on page 223](#).
- The `IT_Certificate::X509CertChain` object that Orbix passes to the `IT_TLS::CertValidator::validate_cert_chain()` operation.
- Using the `IT_Certificate::X509CertificateFactory` interface, which creates an `IT_Certificate::X509Cert` object from DER data.

The certificate can be accessed through the `IT_Certificate::X509Cert` interface. For more information on this interface, see the *CORBA Programmer's Reference*.

Part 6

Appendices

In this part

This part contains the following appendices:

Security Configuration	page 251
ASN.1 and Distinguished Names	page 307
Association Options	page 313
SSL/TLS Sample Configurations	page 317
Security Recommendations	page 323
Action-Role Mapping DTD	page 327

Security Configuration

This appendix describes configuration variables used by the Orbix Security Framework. The security infrastructure is highly configurable.

In this appendix

This appendix discusses the following topics:

Applying Constraints to Certificates	page 253
initial_references	page 255
plugins:atli2_tls	page 256
plugins:csi	page 257
plugins:gsp	page 258
plugins:iiop_tls	page 264
plugins:security	page 269
plugins:systemssl_toolkit	page 270
policies	page 272
policies:csi	page 279
policies:https	page 282

policies:iop_tls	page 289
principal_sponsor	page 300
principal_sponsor:csi	page 304

Applying Constraints to Certificates

Certificate constraints policy

You can use the `CertConstraintsPolicy` to apply constraints to peer X.509 certificates by the default `CertificateValidatorPolicy`. These conditions are applied to the owner's distinguished name (DN) on the first certificate (peer certificate) of the received certificate chain. Distinguished names are made up of a number of distinct fields, the most common being Organization Unit (OU) and Common Name (CN).

Configuration variable

You can specify a list of constraints to be used by `CertConstraintsPolicy` through the `policies:iiop_tls:certificate_constraints_policy` or `policies:https:certificate_constraints_policy` configuration variables. For example:

```
policies:iiop_tls:certificate_constraints_policy =
    ["CN=Johnny*,OU=[unit1|IT_SSL],O=Orbix
     Services,C=Ireland,ST=Dublin,L=Earth", "CN=Paul*,OU=SSLTEAM,O=
     Orbix Services,C=Ireland,ST=Dublin,L=Earth",
     "CN=TheOmnipotentOne"];
```

Constraint language

These are the special characters and their meanings in the constraint list:

*	Matches any text. For example: an* matches ant and anger, but not aunt
[]	Grouping symbols.
	Choice symbol. For example: OU=[unit1 IT_SSL] signifies that if the OU is unit1 or IT_SSL, the certificate is acceptable.
=, !=	Signify equality and inequality respectively.

Example

This is an example list of constraints:

```
policies:iiop_tls:certificate_constraints_policy = [
    "OU=[unit1|IT_SSL],CN=Steve*,L=Dublin",
    "OU=IT_ART*,OU!=IT_ARTtesters,CN=[Jan|Donal],ST=
    Boston" ];
```

This constraint list specifies that a certificate is deemed acceptable if and only if it satisfies one or more of the constraint patterns:

```

If
    The OU is unit1 or IT_SSL
    And
    The CN begins with the text Steve
    And
    The location is Dublin
Then the certificate is acceptable
Else (moving on to the second constraint)
If
    The OU begins with the text IT_ART but isn't IT_ARTtesters
    And
    The common name is either Donal or Jan
    And
    The State is Boston
Then the certificate is acceptable
Otherwise the certificate is unacceptable.

```

The language is like a boolean OR, trying the constraints defined in each line until the certificate satisfies one of the constraints. Only if the certificate fails all constraints is the certificate deemed invalid.

Note that this setting can be sensitive about white space used within it. For example, "CN =" might not be recognized, where "CN=" is recognized.

Distinguished names

For more information on distinguished names, see [Appendix B on page 307](#).

initial_references

The `initial_references` namespace contains the following configuration variables:

- [IT_TLS_Toolkit:plugin](#)

IT_TLS_Toolkit:plugin

This configuration variable enables you to specify the underlying SSL/TLS toolkit to be used by . It is used in conjunction with the `plugins:baltimore_toolkit:shlib_name`, `plugins:schannel_toolkit:shlib_name` (Windows only) and `plugins:systemssl_toolkit:shlib_name` (z/OS only) configuration variables to implement SSL/TLS toolkit replaceability.

The only toolkit supported by Orbix Mainframe is the System SSL toolkit.

For example, to specify that an application should use the System SSL toolkit, you would set configuration variables as follows:

```
initial_references:IT_TLS_Toolkit:plugin = "systemssl_toolkit";
plugins:systemssl_toolkit:shlib_name = "ORXSSSL";
```

plugins:atli2_tls

The `plugins:atli2_tls` namespace contains the following variable:

- `use_jsse_tk`

use_jsse_tk

(Java only) Specifies whether or not to use the JSSE/JCE architecture with . If `true`, uses the JSSE/JCE architecture to implement SSL/TLS security; if `false`, uses the Baltimore SSL/TLS toolkit.

The default is `false`.

plugins:csi

The `policies:csi` namespace includes variables that specify settings for Common Secure Interoperability version 2 (CSIv2):

- `shlib_name`
 - `use_legacy_policies`
-

shlib_name

`shlib_name` identifies the DLL that contains the `csi` plug-in implementation:

```
plugins:csi:shlib_name = "ORXCSIP";
```

The `csi` plug-in becomes associated with the `ORXCSIP` DLL, where `ORXCSIP` is the unversioned or similar word base name of the library.

use_legacy_policies

`use_legacy_policies` is a boolean variable that specifies whether the application can be programmed using the new CSIv2 policy types or the older (legacy) CSIv2 policy types.

If `plugins:csi:use_legacy_policies` is set to `true`, you can program CSIv2 using the following policies:

- `IT_CSI::AuthenticationServicePolicy`
- `IT_CSI::AttributeServicePolicy`

If `plugins:csi:use_legacy_policies` is set to `false`, you can program CSIv2 using the following policies:

- `IT_CSI::AttributeServiceProtocolClient`
- `IT_CSI::AttributeServiceProtocolServer`

Default is `false`.

plugins:gsp

The `plugins:gsp` namespace includes variables that specify settings for the Generic Security Plugin (GSP). This provides authorization by checking a user's roles against the permissions stored in an action-role mapping file. It includes the following:

- `accept_asserted_authorization_info`
- `action_role_mapping_file`
- `assert_authorization_info`
- `authentication_cache_size`
- `authentication_cache_timeout`
- `authorization_realm`
- `enable_authorization`
- `enable_gssup_sso`
- `enable_user_id_logging`
- `enable_x509_sso`
- `enforce_secure_comms_to_sso_server`
- `enable_security_service_cert_authentication`
- `retrieve_isf_auth_principal_info_for_all_realms`
- `sso_server_certificate_constraints`
- `use_client_load_balancing`

accept_asserted_authorization_info

If `false`, SAML data is not read from incoming connections. Default is `true`.

action_role_mapping_file

Specifies the action-role mapping file URL. For example:

```
plugins:gsp:action_role_mapping_file =  
    "file:///my/action/role/mapping";
```

assert_authorization_info

If `false`, SAML data is not sent on outgoing connections. Default is `true`.

authentication_cache_size

The maximum number of credentials stored in the authentication cache. If this size is exceeded the oldest credential in the cache is removed.

A value of `-1` (the default) means unlimited size. A value of `0` means disable the cache.

authentication_cache_timeout

The time (in seconds) after which a credential is considered *stale*. Stale credentials are removed from the cache and the server must re-authenticate with the security service on the next call from that user. The cache timeout should be configured to be smaller than the timeout set in the

`is2.properties` file (by default, that setting is `is2.sso.session.timeout=600`).

A value of `-1` (the default) means an infinite time-out. A value of `0` means disable the cache.

authorization_realm

`authorization_realm` specifies the iSF authorization realm to which a server belongs. The value of this variable determines which of a user's roles are considered when making an access control decision.

For example, consider a user that belongs to the `ejb-developer` and `corba-developer` roles within the `Engineering` realm, and to the `ordinary` role within the `Sales` realm. If you set `plugins:gsp:authorization_realm` to `Sales` for a particular server, only the `ordinary` role is considered when making access control decisions (using the `action-role` mapping file).

enable_authorization

A boolean GSP policy that, when `true`, enables authorization using action-role mapping ACLs in server.

Default is `true`.

enable_gssup_sso

Enables SSO with a username and a password (that is, GSSUP) when set to `true`.

enable_user_id_logging

A boolean variable that enables logging of user IDs on the server side. Default is `false`.

Up until the release of Orbix 6.1 SP1, the GSP plug-in would log messages containing user IDs. For example:

```
[junit] Fri, 28 May 2004 12:17:22.0000000 [SLEEPY:3284]
      (IT_CSI:205) I - User alice authenticated successfully.
```

In some cases, however, it might not be appropriate to expose user IDs in the Orbix log. From Orbix 6.2 onward, the default behavior of the GSP plug-in is changed, so that user IDs are *not* logged by default. To restore the pre-Orbix 6.2 behavior and log user IDs, set this variable to `true`.

enable_x509_sso

Enables certificate-based SSO when set to `true`.

enforce_secure_comms_to_sso_server

Enforces a secure SSL/TLS link between a client and the login service when set to `true`. When this setting is true, the value of the SSL/TLS client secure invocation policy does *not* affect the connection between the client and the login service.

Default is `true`.

enable_security_service_cert_authentication

A boolean GSP policy that enables X.509 certificate-based authentication on the server side using the `security` service.

Default is `false`.

retrieve_isf_auth_principal_info_for_all_realms

A boolean setting that determines whether the GSP plug-in retrieves role and realm data for all realms, when authenticating user credentials. If `true`, the GSP plug-in retrieves the user's role and realm data for all realms; if `false`, the GSP plug-in retrieves the user's role and realm data only for the realm specified by `plugins:gsp:authorization_realm`.

Setting this variable to `false` can provide a useful performance optimization in some applications. But you must take special care to configure the application correctly for making operation invocations between different realms.

Default is `true`.

sso_server_certificate_constraints

A special certificate constraints policy that applies *only* to the SSL/TLS connection between the client and the SSO login server. For details of the pattern constraint language, see [“Applying Constraints to Certificates” on page 253](#).

use_client_load_balancing

A boolean variable that enables load balancing over a cluster of security services. If an application is deployed in a domain that uses security service clustering, the application should be configured to use *client load balancing* (in this context, *client* means a client of the `security` service). See also `policies:iiop_tls:load_balancing_mechanism`.

Default is `true`.

plugins:https

The `plugins:https` namespace contains the following variable:

- [ClassName](#)

ClassName

(Java only) This variable specifies the class name of the `https` plug-in implementation. For example:

```
plugins:https:ClassName = "com.ionacorba.https.HTTPSPlugIn";
```

plugins:iiop_tls

The `plugins:iiop_tls` namespace contains the following variables:

- `buffer_pool:recycle_segments`
- `buffer_pool:segment_preallocation`
- `buffer_pools:max_incoming_buffers_in_pool`
- `buffer_pools:max_outgoing_buffers_in_pool`
- `cert_expiration_warning_days`
- `delay_credential_gathering_until_handshake`
- `enable_iiop_1_0_client_support`
- `enable_warning_for_approaching_cert_expiration`
- `incoming_connections:hard_limit`
- `incoming_connections:soft_limit`
- `outgoing_connections:hard_limit`
- `outgoing_connections:soft_limit`
- `own_credentials_warning_cert_constraints`
- `tcp_listener:reincarnate_attempts`
- `tcp_listener:reincarnation_retry_backoff_ratio`
- `tcp_listener:reincarnation_retry_delay`

buffer_pool:recycle_segments

(Java only) When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:buffer_pool:recycle_segments` variable's value.

buffer_pool:segment_preallocation

(Java only) When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:buffer_pool:segment_preallocation` variable's value.

buffer_pools:max_incoming_buffers_in_pool

(C++ only) When this variable is set, the `iop_tls` plug-in reads this variable's value instead of the

`plugins:iop:buffer_pools:max_incoming_buffers_in_pool` variable's value.

buffer_pools:max_outgoing_buffers_in_pool

(C++ only) When this variable is set, the `iop_tls` plug-in reads this variable's value instead of the

`plugins:iop:buffer_pools:max_outgoing_buffers_in_pool` variable's value.

cert_expiration_warning_days

(*Since Orbix 6.2 SPI*) Specifies the threshold for the number of days left to certificate expiration, before Orbix issues a warning. If the application's own certificate is due to expire in less than the specified number of days, Orbix issues a warning message to the log.

Default is 31 days.

See also the following related configuration variables:

`plugins:iop_tls:enable_warning_for_approaching_cert_expiration`
`plugins:iop_tls:own_credentials_warning_cert_constraints`

delay_credential_gathering_until_handshake

(Windows and Schannel only) This client configuration variable provides an alternative to using the `principal_sponsor` variables to specify an application's own certificate. When this variable is set to `true` and `principal_sponsor:use_principal_sponsor` is set to `false`, the client delays sending its certificate to a server. The client will wait until the server *explicitly* requests the client to send its credentials during the SSL/TLS handshake.

This configuration variable can be used in conjunction with the `plugins:schannel:prompt_with_credential_choice` configuration variable.

enable_iiop_1_0_client_support

This variable enables client-side interoperability of SSL/TLS applications with legacy IIOp 1.0 SSL/TLS servers, which do not support IIOp 1.1.

The default value is `false`. When set to `true`, SSL/TLS searches secure target IIOp 1.0 object references for legacy IIOp 1.0 SSL/TLS tagged component data, and attempts to connect on the specified port.

Note: This variable will not be necessary for most users.

enable_warning_for_approaching_cert_expiration

(*Since Orbix 6.2 SPI*) Enables warnings to be sent to the log, if an application's own certificate is imminently about to expire. The boolean value can have the following values: `true`, enables the warning feature; `false`, disables the warning feature.

Default is `true`.

See also the following related configuration variables:

```
plugins:iiop_tls:cert_expiration_warning_days
plugins:iiop_tls:own_credentials_warning_cert_constraints
```

incoming_connections:hard_limit

Specifies the maximum number of incoming (server-side) connections permitted to IIOp. IIOp does not accept new connections above this limit. Defaults to -1 (disabled).

When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:incoming_connections:hard_limit` variable's value.

Please see the chapter on ACM in the *CORBA Programmer's Guide* for further details.

incoming_connections:soft_limit

Specifies the number of connections at which IIOp should begin closing incoming (server-side) connections. Defaults to -1 (disabled).

When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:incoming_connections:soft_limit` variable's value.

Please see the chapter on ACM in the *CORBA Programmer's Guide* for further details.

outgoing_connections:hard_limit

When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:outgoing_connections:hard_limit` variable's value.

outgoing_connections:soft_limit

When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:outgoing_connections:soft_limit` variable's value.

own_credentials_warning_cert_constraints

(Since Orbix 6.2 SP1) Set this certificate constraints variable, if you would like to avoid deploying certain certificates as an own certificate. A warning is issued, if the own certificate's subject DN matches the constraints specified by this variable (see [“Applying Constraints to Certificates” on page 253](#) for details of the constraint language). For example, you might want to generate a warning in case you accidentally deployed a Micro Focus demonstration certificate.

Default is an empty list, [].

Note: This warning is *not* related to certificate expiration and works independently of the certificate expiration warning.

tcp_listener:reincarnate_attempts

(Windows only)

`plugins:iioptls:tcp_listener:reincarnate_attempts` specifies the number of times that a Listener recreates its listener socket after receiving a `SocketException`.

Sometimes a network error may occur, which results in a listening socket being closed. On Windows, you can configure the listener to attempt a reincarnation, which enables new connections to be established. This variable only affects Java and C++ applications on Windows. Defaults to 0 (no attempts).

tcp_listener:reincarnation_retry_backoff_ratio

(Windows only)

`plugins:iioptls:tcp_listener:reincarnation_retry_delay` specifies a delay between reincarnation attempts. Data type is `long`. Defaults to 0 (no delay).

tcp_listener:reincarnation_retry_delay

(Windows only)

`plugins:iioptls:tcp_listener:reincarnation_retry_backoff_ratio` specifies the degree to which delays between retries increase from one retry to the next. Datatype is `long`. Defaults to 1.

plugins:security

The `plugins:security` namespace contains the following variable:

- [share_credentials_across_orbs](#)

share_credentials_across_orbs

Enables own security credentials to be shared across ORBs. Normally, when you specify an own SSL/TLS credential (using the principal sponsor or the principal authenticator), the credential is available only to the ORB that created it. By setting the

`plugins:security:share_credentials_across_orbs` variable to `true`, however, the own SSL/TLS credentials created by one ORB are automatically made available to any other ORBs that are configured to share credentials.

See also `principal_sponsor:csi:use_existing_credentials` for details of how to enable sharing of CSI credentials.

Default is `false`.

plugins:systemssl_toolkit

The `plugins:systemssl_toolkit` namespace contains the following variables:

- `hfs_keyring_file_password`
- `hfs_keyring_file_stashfile`
- `hfs_keyring_filename`
- `saf_keyring`
- `session_cache_size`
- `session_cache_validity_period`

hfs_keyring_file_password

`hfs_keyring_file_password` specifies the password that accesses the key database specified by `plugins:systemssl_toolkit:hfs_keyring_filename`.

hfs_keyring_file_stashfile

`hfs_keyring_file_stashfile` specifies the name of a stash file containing the password that accesses the key database specified by `plugins:systemssl_toolkit:hfs_keyring_filename`. The stash file stores the password in encrypted form.

Note: Either `hfs_keyring_file_password` or `hfs_keyring_file_stashfile` can be used to specify the password, but not both.

hfs_keyring_filename

`hfs_keyring_filename` specifies the name of a key ring file (database of keys) within a hierarchical file system. For example, to specify the `/keyring/key.kdb` key ring file:

```
plugins:systemssl_toolkit:hfs_keyring_filename =  
    "/keyring/key.kdb";
```

saf_keyring

`saf_keyring` specifies the name of an SAF key ring (for example, an RACF key ring) from which an application retrieves authentication data. For example, to use the SAF key ring named `TESTRING`:

```
plugins:systemssl_toolkit:saf_keyring = "TESTRING";
```

session_cache_size

`session_cache_size` specifies the size of the System SSL session identifier cache. The oldest entry will be removed when the cache is full, in order to make space for a new entry. The range is 0-64000 and defaults to 512. Session identifiers will not be remembered if a value of 0 is specified.

session_cache_validity_period

`session_cache_validity_period` specifies the number of seconds until an SSL session identifier expires. The range is 0-86400 and defaults to 86400 (one day). System SSL will remember session identifiers for this amount of time. This reduces the amount of data exchanged during the SSL handshake when a complete initial handshake has already been performed. Session identifiers will not be remembered if a value of 0 is specified.

policies

The `policies` namespace defines the default CORBA policies for an ORB. Many of these policies can also be set programmatically from within an application. SSL/TLS-specific variables in the `policies` namespace include:

- `allow_unauthenticated_clients_policy`
- `certificate_constraints_policy`
- `client_secure_invocation_policy:requires`
- `client_secure_invocation_policy:supports`
- `max_chain_length_policy`
- `mechanism_policy:accept_v2_hellos`
- `mechanism_policy:ciphersuites`
- `mechanism_policy:protocol_version`
- `target_secure_invocation_policy:requires`
- `target_secure_invocation_policy:supports`
- `trusted_ca_list_policy`

allow_unauthenticated_clients_policy

(Deprecated in favor of

`policies:iiop_tls:allow_unauthenticated_clients_policy` and `policies:https:allow_unauthenticated_clients_policy`.)

A generic variable that sets this policy both for `iiop_tls` and `https`. The recommended alternative is to use the variables prefixed by `policies:iiop_tls` and `policies:https` instead, which take precedence over this generic variable.

certificate_constraints_policy

(Deprecated in favor of

`policies:iiop_tls:certificate_constraints_policy` and
`policies:https:certificate_constraints_policy`.)

A generic variable that sets this policy both for `iiop_tls` and `https`. The recommended alternative is to use the variables prefixed by `policies:iiop_tls` and `policies:https` instead, which take precedence over this generic variable.

client_secure_invocation_policy:requires

(Deprecated in favor of

`policies:iiop_tls:client_secure_invocation_policy:requires` and
`policies:https:client_secure_invocation_policy:requires`.)

A generic variable that sets this policy both for `iiop_tls` and `https`. The recommended alternative is to use the variables prefixed by `policies:iiop_tls` and `policies:https` instead, which take precedence over this generic variable.

client_secure_invocation_policy:supports

(Deprecated in favor of

`policies:iiop_tls:client_secure_invocation_policy:supports` and
`policies:https:client_secure_invocation_policy:supports`.)

A generic variable that sets this policy both for `iiop_tls` and `https`. The recommended alternative is to use the variables prefixed by `policies:iiop_tls` and `policies:https` instead, which take precedence over this generic variable.

max_chain_length_policy

(Deprecated in favor of `policies:iiop_tls:max_chain_length_policy` and `policies:https:max_chain_length_policy`.)

`max_chain_length_policy` specifies the maximum certificate chain length that an ORB will accept. The policy can also be set programmatically using the `IT_TLS_API::MaxChainLengthPolicy` CORBA policy. Default is 2.

Note: The `max_chain_length_policy` is not currently supported on the z/OS platform.

mechanism_policy:accept_v2_hellos

(See also `policies:iiop_tls:mechanism_policy:accept_v2_hellos` and `policies:https:mechanism_policy:accept_v2_hellos`.)

The `accept_v2_hellos` policy is a special setting that facilitates interoperability with older deployments of Orbix on z/OS. When `true`, the application accepts V2 client hellos, but continues the handshake using either the `SSL_V3` or `TLS_V1` protocol. When `false`, the application throws an error, if it receives a V2 client hello. The default is `false`. For example:

```
policies:mechanism_policy:accept_v2_hellos = "true";
```

mechanism_policy:ciphersuites

(Deprecated in favor of

`policies:iiop_tls:mechanism_policy:ciphersuites` and `policies:https:mechanism_policy:ciphersuites`.)

`mechanism_policy:ciphersuites` specifies a list of cipher suites for the default mechanism policy. One or more of the cipher suites listed below can be specified in this list:

TLS_RSA_WITH_NULL_MD5
TLS_RSA_WITH_NULL_SHA
TLS_RSA_EXPORT_WITH_RC4_40_MD5

TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_DSS_WITH_AES_128_GCM_SHA256
TLS_DHE_DSS_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_RC4_128_SHA
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA

TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_AES_256_GCM_SHA384
TLS_AES_128_GCM_SHA256
TLS_CHACHA20_POLY1305_SHA256

mechanism_policy:protocol_version

(Deprecated in favor of

`policies:iioptls:mechanism_policy:protocol_version` and `policies:https:mechanism_policy:protocol_version`.)

`mechanism_policy:protocol_version` specifies the list of protocol versions used by a security capsule (ORB instance). The list can include one or more of the following values:

- SSL_V3
- TLS_V1
- TLS_V1_1
- TLS_V1_2
- TLS_V1_3

For example:

```
policies:mechanism_policy:protocol_version=["TLS_V1", "SSL_V3"];
```

target_secure_invocation_policy:requires

(Deprecated in favor of

`policies:iiop_tls:target_secure_invocation_policy:requires` and
`policies:https:target_secure_invocation_policy:requires`.)

`target_secure_invocation_policy:requires` specifies the minimum level of security required by a server. The value of this variable is specified as a list of association options.

Note: In accordance with CORBA security, this policy cannot be downgraded programmatically by the application.

target_secure_invocation_policy:supports

(Deprecated in favor of

`policies:iiop_tls:target_secure_invocation_policy:supports` and
`policies:https:target_secure_invocation_policy:supports`.)

`supports` specifies the maximum level of security supported by a server. The value of this variable is specified as a list of association options. This policy can be upgraded programmatically using either the `QOP` or the `EstablishTrust` policies.

trusted_ca_list_policy

(Deprecated in favor of `policies:iiop_tls:trusted_ca_list_policy` and `policies:https:trusted_ca_list_policy`.)

`trusted_ca_list_policy` specifies a list of filenames, each of which contains a concatenated list of CA certificates in PEM format. The aggregate of the CAs in all of the listed files is the set of trusted CAs.

For example, you might specify two files containing CA lists as follows:

```
policies:trusted_ca_list_policy =  
  ["install_dir/asp/version/etc/tls/x509/ca/ca_list1.pem",  
   "install_dir/asp/version/etc/tls/x509/ca/ca_list_extra.pem"];
```

The purpose of having more than one file containing a CA list is for administrative convenience. It enables you to group CAs into different lists and to select a particular set of CAs for a security domain by choosing the appropriate CA lists.

Note: The `trusted_ca_list_policy` configuration variable is not used with System SSL on the z/OS platform. The System SSL toolkit obtains its CA list from the underlying SSL repository (the SAF key ring or the HFS key database).

policies:csi

The `policies:csi` namespace includes variables that specify settings for Common Secure Interoperability version 2 (CSlv2):

- `attribute_service:backward_trust:enabled`
- `attribute_service:client_supports`
- `attribute_service:target_supports`
- `auth_over_transport:authentication_service`
- `auth_over_transport:client_supports`
- `auth_over_transport:server_domain_name`
- `auth_over_transport:target_requires`
- `auth_over_transport:target_supports`

`attribute_service:backward_trust:enabled`

(Obsolete)

`attribute_service:client_supports`

`attribute_service:client_supports` is a client-side policy that specifies the association options supported by the CSlv2 attribute service (principal propagation). The only association option that can be specified is `IdentityAssertion`. This policy is normally specified in an intermediate server so that it propagates CSlv2 identity tokens to a target server. For example:

```
policies:csi:attribute_service:client_supports =  
    ["IdentityAssertion"];
```

attribute_service:target_supports

`attribute_service:target_supports` is a server-side policy that specifies the association options supported by the CSIV2 attribute service (principal propagation). The only association option that can be specified is `IdentityAssertion`. For example:

```
policies:csi:attribute_service:target_supports =  
    ["IdentityAssertion"];
```

auth_over_transport:authentication_service

(Java CSI plug-in only) The name of a Java class that implements the `IT_CSI::AuthenticateGSSUPCredentials` IDL interface. The authentication service is implemented as a callback object that plugs into the CSIV2 framework on the server side. By replacing this class with a custom implementation, you could potentially implement a new security technology domain for CSIV2.

By default, if no value for this variable is specified, the Java CSI plug-in uses a default authentication object that always returns `false` when the `authenticate()` operation is called.

auth_over_transport:client_supports

`auth_over_transport:client_supports` is a client-side policy that specifies the association options supported by CSIV2 authorization over transport. The only association option that can be specified is `EstablishTrustInClient`. For example:

```
policies:csi:auth_over_transport:client_supports =  
    ["EstablishTrustInClient"];
```

auth_over_transport:server_domain_name

The iSF security domain (CSlv2 authentication domain) to which this server application belongs. The iSF security domains are administered within an overall security technology domain.

The value of the `server_domain_name` variable will be embedded in the IORs generated by the server. A CSlv2 client about to open a connection to this server would check that the domain name in its own CSlv2 credentials matches the domain name embedded in the IOR.

auth_over_transport:target_requires

`auth_over_transport:target_requires` is a server-side policy that specifies the association options required for CSlv2 authorization over transport. The only association option that can be specified is `EstablishTrustInClient`. For example:

```
policies:csi:auth_over_transport:target_requires =  
  ["EstablishTrustInClient"];
```

auth_over_transport:target_supports

`auth_over_transport:target_supports` is a server-side policy that specifies the association options supported by CSlv2 authorization over transport. The only association option that can be specified is `EstablishTrustInClient`. For example:

```
policies:csi:auth_over_transport:target_supports =  
  ["EstablishTrustInClient"];
```

policies:https

The `policies:https` namespace contains variables used to configure the https plugin. It contains the following variables:

- `allow_unauthenticated_clients_policy`
- `browser_navigation:enabled`
- `certificate_constraints_policy`
- `client_secure_invocation_policy:requires`
- `client_secure_invocation_policy:supports`
- `max_chain_length_policy`
- `mechanism_policy:accept_v2_hellos`
- `mechanism_policy:ciphersuites`
- `mechanism_policy:protocol_version`
- `send_timeout`
- `session_caching_policy`
- `target_secure_invocation_policy:requires`
- `target_secure_invocation_policy:supports`
- `trace_requests:enabled`
- `trusted_ca_list_policy`

allow_unauthenticated_clients_policy

A boolean variable that specifies whether a server will allow a client to establish a secure connection without sending a certificate. Default is `false`.

This configuration variable is applicable *only* in the special case where the target secure invocation policy is set to require `NoProtection` (a semi-secure server).

browser_navigation:enabled

Specifies whether you can use the browser interface to drill down to the list of available Web service endpoints. The default value is `true`, which means you can enter a high-level URL (for example, `https://host:port`), and click through subsequent screens to view to the list of available services and the associated WSDL.

certificate_constraints_policy

A list of constraints applied to peer certificates—see [“Applying Constraints to Certificates” on page 253](#) for the syntax of the pattern constraint language. If a peer certificate fails to match any of the constraints, the certificate validation step will fail.

The policy can also be set programmatically using the `IT_TLS_API::CertConstraintsPolicy` CORBA policy. Default is no constraints.

client_secure_invocation_policy:requires

Specifies the minimum level of security required by a client. The value of this variable is specified as a list of association options—see the *Orbix Security Guide* for details on how to set SSL/TLS association options.

Note: In accordance with CORBA security, this policy cannot be downgraded programmatically by the application.

client_secure_invocation_policy:supports

Specifies the initial maximum level of security supported by a client. The value of this variable is specified as a list of association options—see the *Orbix Security Guide* for details on how to set SSL/TLS association options.

Note: This policy can be upgraded programmatically using either the `QOP` or the `EstablishTrust` policies.

max_chain_length_policy

The maximum certificate chain length that an ORB will accept (see the discussion of certificate chaining in the *Orbix Security Guide*).

The policy can also be set programmatically using the `IT_TLS_API::MaxChainLengthPolicy` CORBA policy. Default is 2.

Note: The `max_chain_length_policy` is not currently supported on the z/OS platform.

mechanism_policy:accept_v2_hellos

This HTTPS-specific policy overrides the generic `policies:mechanism_policy:accept_v2_hellos` policy.

The `accept_v2_hellos` policy is a special setting that facilitates HTTPS interoperability with certain Web browsers. Many Web browsers send SSL V2 client hellos, because they do not know what SSL version the server supports.

When `true`, the server accepts V2 client hellos, but continues the handshake using either the `SSL_V3` or `TLS_V1` protocol. When `false`, the server throws an error, if it receives a V2 client hello. The default is `true`.

Note: This default value is deliberately different from the `policies:iiop_tls:mechanism_policy:accept_v2_hellos` default value.

For example:

```
policies:https:mechanism_policy:accept_v2_hellos = "true";
```

mechanism_policy:ciphersuites

Specifies a list of cipher suites for the default mechanism policy. One or more of the following cipher suites can be specified in this list:

<code>TLS_RSA_WITH_NULL_MD5</code>
<code>TLS_RSA_WITH_NULL_SHA</code>

TLS_RSA_EXPORT_WITH_RC4_40_MD5
TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_DSS_WITH_AES_128_GCM_SHA256
TLS_DHE_DSS_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_RC4_128_SHA
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA

TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_AES_256_GCM_SHA384
TLS_AES_128_GCM_SHA256
TLS_CHACHA20_POLY1305_SHA256

mechanism_policy:protocol_version

This HTTPS-specific policy overrides the generic `policies:mechanism_policy:protocol_version` policy.

Specifies the list of protocol versions used by a security capsule (ORB instance). The list can include one or more of:

- `SSL_V3`
- `TLS_V1`
- `TLS_V1_1`
- `TLS_V1_2`
- `TLS_V1_3`

The default setting is `TLS_V1` and `TLS_V1_3`.

For example:

```
policies:https:mechanism_policy:protocol_version = ["TLS_V1",
"SSL_V3"];
```

send_timeout

Enables you to abort an HTTPS send reply attempt to the target Web service consumer if this expiry setting times out. The value is expressed in milliseconds. This setting relates to the time taken to send the entire HTTP message to the remote Web service peer.

```
policies:https:send_timeout = "5000";
```

session_caching_policy

When this policy is set, the `https` plug-in reads this policy's value instead of the [policies:session_caching](#) policy's value (C++) or [policies:session_caching_policy](#) policy's value (Java).

target_secure_invocation_policy:requires

Specifies the minimum level of security required by a server. The value of this variable is specified as a list of association options—see the *Orbix Security Guide* for more details about association options.

In accordance with CORBA security, this policy cannot be downgraded programmatically by the application.

target_secure_invocation_policy:supports

Specifies the maximum level of security supported by a server. The value of this variable is specified as a list of association options—see the *Orbix Security Guide* for more details about association options.

This policy can be upgraded programmatically using either the `QOP` or the `EstablishTrust` policies.

trace_requests:enabled

Specifies whether the contents of each HTTPS message should be sent to the event-log stream as `INFO` messages. If no value is specified for this variable, it defaults to `false`, and no `INFO` messages are sent to the event log stream.

trusted_ca_list_policy

Contains a list of filenames (or a single filename), each of which contains a concatenated list of CA certificates in PEM format. The aggregate of the CAs in all of the listed files is the set of trusted CAs.

For example, you might specify two files containing CA lists as follows:

```
policies:trusted_ca_list_policy =  
    ["ASPInstallDir/asp/6.3/etc/tls/x509/ca/ca_list1.pem",  
     "ASPInstallDir/asp/6.3/etc/tls/x509/ca/ca_list_extra.pem"];
```

The purpose of having more than one file containing a CA list is for administrative convenience. It enables you to group CAs into different lists and to select a particular set of CAs for a security domain by choosing the appropriate CA lists.

Note: The `trusted_ca_list_policy` configuration variable is not used with System SSL on the z/OS platform. The System SSL toolkit obtains its CA list from the underlying SSL repository (the SAF key ring or the HFS key database).

policies:iiop_tls

The `policies:iiop_tls` namespace contains variables used to set IIOP-related policies for a secure environment. These settings affect the `iiop_tls` plugin. It contains the following variables:

- `buffer_sizes_policy:default_buffer_size`
- `buffer_sizes_policy:max_buffer_size`
- `certificate_constraints_policy`
- `client_secure_invocation_policy:requires`
- `client_secure_invocation_policy:supports`
- `client_version_policy`
- `connection_attempts`
- `connection_retry_delay`
- `load_balancing_mechanism`
- `max_chain_length_policy`
- `mechanism_policy:accept_v2_hellos`
- `mechanism_policy:ciphersuites`
- `mechanism_policy:protocol_version`
- `server_address_mode_policy:local_domain`
- `server_address_mode_policy:local_hostname`
- `server_address_mode_policy:port_range`
- `server_address_mode_policy:publish_hostname`
- `server_version_policy`
- `target_secure_invocation_policy:requires`
- `target_secure_invocation_policy:supports`
- `tcp_options_policy:no_delay`
- `tcp_options_policy:recv_buffer_size`
- `tcp_options_policy:send_buffer_size`
- `trusted_ca_list_policy`

buffer_sizes_policy:default_buffer_size

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:iiop:buffer_sizes_policy:default_buffer_size` policy's value.

`buffer_sizes_policy:default_buffer_size` specifies, in bytes, the initial size of the buffers allocated by IIOp. Defaults to 16000. This value must be greater than 80 bytes, and must be evenly divisible by 8.

buffer_sizes_policy:max_buffer_size

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:iiop:buffer_sizes_policy:max_buffer_size` policy's value.

`buffer_sizes_policy:max_buffer_size` specifies the maximum buffer size permitted by IIOp, in kilobytes. Defaults to 512. A value of -1 indicates unlimited size. If not unlimited, this value must be greater than 80.

certificate_constraints_policy

A list of constraints applied to peer certificates—see the discussion of certificate constraints in the [security guide](#) for the syntax of the pattern constraint language. If a peer certificate fails to match any of the constraints, the certificate validation step will fail.

The policy can also be set programmatically using the `IT_TLS_API::CertConstraintsPolicy` CORBA policy. Default is no constraints.

client_secure_invocation_policy:requires

Specifies the minimum level of security required by a client. The value of this variable is specified as a list of association options—see the *Security Guide* for more details about association options.

In accordance with CORBA security, this policy cannot be downgraded programmatically by the application.

client_secure_invocation_policy:supports

Specifies the initial maximum level of security supported by a client. The value of this variable is specified as a list of association options—see the *Security Guide* for more details about association options.

This policy can be upgraded programmatically using either the `QOP` or the `EstablishTrust` policies.

client_version_policy

`client_version_policy` specifies the highest IIOp version used by clients. A client uses the version of IIOp specified by this variable, or the version specified in the IOR profile, whichever is lower. Valid values for this variable are: 1.0, 1.1, and 1.2.

For example, the following file-based configuration entry sets the server IIOp version to 1.1.

```
policies:iop:server_version_policy="1.1";
```

The following `itadmin` command set this variable:

```
itadmin variable modify -type string -value "1.1"  
policies:iop:server_version_policy
```

connection_attempts

`connection_attempts` specifies the number of connection attempts used when creating a connected socket using a Java application. Defaults to 5.

connection_retry_delay

`connection_retry_delay` specifies the delay, in seconds, between connection attempts when using a Java application. Defaults to 2.

load_balancing_mechanism

Specifies the load balancing mechanism for the client of a security service cluster (see also `plugins:gsp:use_client_load_balancing`). In this context, a client can also be an server. This policy only affects connections made using IORs that contain multiple addresses. The `iiop_tls` plug-in load balances over the addresses embedded in the IOR.

The following mechanisms are supported:

- `random`—choose one of the addresses embedded in the IOR at random (this is the default).
- `sequential`—choose the first address embedded in the IOR, moving on to the next address in the list only if the previous address could not be reached.

max_chain_length_policy

This policy overrides `policies:max_chain_length_policy` for the `iiop_tls` plugin.

The maximum certificate chain length that an ORB will accept.

The policy can also be set programmatically using the `IT_TLS_API::MaxChainLengthPolicy` CORBA policy. Default is 2.

Note: The `max_chain_length_policy` is not currently supported on the z/OS platform.

mechanism_policy:accept_v2_hellos

This IIOp/TLS-specific policy overrides the generic `policies:mechanism_policy:accept_v2_hellos` policy.

The `accept_v2_hellos` policy is a special setting that facilitates interoperability with older deployments of Orbix on z/OS.

security on the z/OS platform is based on IBM's System/SSL toolkit, which implements SSL version 3, but does so by using SSL version 2 hellos as part of the handshake. This form of handshake causes interoperability problems, because applications on other platforms identify the handshake as an SSL

version 2 handshake. The misidentification of the SSL protocol version can be avoided by setting the `accept_v2_hellos` policy to `true` in the non-z/OS application (this bug also affects some old versions of Microsoft Internet Explorer).

When `true`, the application accepts V2 client hellos, but continues the handshake using either the `SSL_V3` or `TLS_V1` protocol. When `false`, the application throws an error, if it receives a V2 client hello. The default is `false`.

Note: This default value is deliberately different from the `policies:https:mechanism_policy:accept_v2_hellos` default value.

For example:

```
policies:iiop_tls:mechanism_policy:accept_v2_hellos = "true";
```

mechanism_policy:ciphersuites

This policy overrides `policies:mechanism_policy:ciphersuites` for the `iiop_tls` plugin.

Specifies a list of cipher suites for the default mechanism policy. One or more of the following cipher suites can be specified in this list:

TLS_RSA_WITH_NULL_MD5
TLS_RSA_WITH_NULL_SHA
TLS_RSA_EXPORT_WITH_RC4_40_MD5
TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA256

TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_DSS_WITH_AES_128_GCM_SHA256
TLS_DHE_DSS_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_RC4_128_SHA
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_AES_256_GCM_SHA384
TLS_AES_128_GCM_SHA256
TLS_CHACHA20_POLY1305_SHA256

mechanism_policy:protocol_version

This IIOP/TLS-specific policy overrides the generic `policies:mechanism_policy:protocol_version` policy.

Specifies the list of protocol versions used by a security capsule (ORB instance). The list can include one or more of:

- SSL_V3
- TLS_V1
- TLS_V1_1
- TLS_V1_2
- TLS_V1_3

The default setting is `TLS_V1` and `TLS_V1_3`.

For example:

```
policies:iiop_tls:mechanism_policy:protocol_version = ["TLS_V1",
"SSL_V3"];
```

The `SSL_V2V3` value is now *deprecated*. It was previously used to facilitate interoperability with applications deployed on the z/OS platform. If you have any legacy configuration that uses `SSL_V2V3`, you should replace it with the following combination of settings:

```
policies:iiop_tls:mechanism_policy:protocol_version = ["SSL_V3",
"TLS_V1"];
policies:iiop_tls:mechanism_policy:accept_v2_hellos = "true";
```

server_address_mode_policy:local_domain

(Java only) When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the

`policies:iiop:server_address_mode_policy:local_domain` policy's value.

server_address_mode_policy:local_hostname

(Java only) When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:iiop:server_address_mode_policy:local_hostname` policy's value.

`server_address_mode_policy:local_hostname` specifies the hostname advertised by the locator daemon, and listened on by server-side IIOP.

Some machines have multiple hostnames or IP addresses (for example, those using multiple DNS aliases or multiple network cards). These machines are often termed *multi-homed hosts*. The `local_hostname` variable supports these type of machines by enabling you to explicitly specify the host that servers listen on and publish in their IORs.

For example, if you have a machine with two network addresses (207.45.52.34 and 207.45.52.35), you can explicitly set this variable to either address:

```
policies:iiop:server_address_mode_policy:local_hostname =  
  "207.45.52.34";
```

By default, the `local_hostname` variable is unspecified. Servers use the default hostname configured for the machine with the Orbix configuration tool.

server_address_mode_policy:port_range

(Java only) When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the

`policies:iiop:server_address_mode_policy:port_range` policy's value.

`server_address_mode_policy:port_range` specifies the range of ports that a server uses when there is no well-known addressing policy specified for the port.

server_address_mode_policy:publish_hostname

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the

`policies:iiop:server_address_mode_policy:publish_hostname` policy's value.

`server_address_mode-policy:publish_hostname` specifies whether IIOP exports hostnames or IP addresses in published profiles. Defaults to `false` (exports IP addresses, and does not export hostnames). To use hostnames in object references, set this variable to `true`, as in the following file-based configuration entry:

```
policies:iiop:server_address_mode_policy:publish_hostname=true
```

The following `itadmin` command is equivalent:

```
itadmin variable create -type bool -value true
policies:iiop:server_address_mode_policy:publish_hostname
```

server_version_policy

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:iiop:server_version_policy` policy's value.

`server_version_policy` specifies the GIOP version published in IIOP profiles. This variable takes a value of either `1.1` or `1.2`. Orbix servers do not publish IIOP 1.0 profiles. The default value is `1.2`.

target_secure_invocation_policy:requires

This policy overrides

`policies:target_secure_invocation_policy:requires` for the `iiop_tls` plugin.

Specifies the minimum level of security required by a server. The value of this variable is specified as a list of association options—see the *Security Guide* for more details about association options.

In accordance with CORBA security, this policy cannot be downgraded programmatically by the application.

target_secure_invocation_policy:supports

This policy overrides

`policies:target_secure_invocation_policy:supports` for the `iiop_tls` plugin.

Specifies the maximum level of security supported by a server. The value of this variable is specified as a list of association options—see the *Security Guide* for more details about association options.

This policy can be upgraded programmatically using either the `QOP` or the `EstablishTrust` policies.

tcp_options_policy:no_delay

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:iiop:tcp_options_policy:no_delay` policy's value.

`tcp_options_policy:no_delay` specifies whether the `TCP_NODELAY` option should be set on connections. Defaults to `false`.

tcp_options_policy:recv_buffer_size

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:iiop:tcp_options_policy:recv_buffer_size` policy's value.

`tcp_options_policy:recv_buffer_size` specifies the size of the TCP receive buffer. This variable can only be set to 0, which corresponds to using the default size defined by the operating system.

tcp_options_policy:send_buffer_size

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:iiop:tcp_options_policy:send_buffer_size` policy's value.

`tcp_options_policy:send_buffer_size` specifies the size of the TCP send buffer. This variable can only be set to 0, which corresponds to using the default size defined by the operating system.

trusted_ca_list_policy

This policy overrides the `policies:trusted_ca_list_policy` for the `iiop_tls` plugin.

Contains a list of filenames (or a single filename), each of which contains a concatenated list of CA certificates in PEM format. The aggregate of the CAs in all of the listed files is the set of trusted CAs.

For example, you might specify two files containing CA lists as follows:

```
policies:trusted_ca_list_policy =  
    ["ASPIInstallDir/asp/6.3/etc/tls/x509/ca/ca_list1.pem",  
     "ASPIInstallDir/asp/6.3/etc/tls/x509/ca/ca_list_extra.pem"];
```

The purpose of having more than one file containing a CA list is for administrative convenience. It enables you to group CAs into different lists and to select a particular set of CAs for a security domain by choosing the appropriate CA lists.

Note: The `trusted_ca_list_policy` configuration variable is not used with System SSL on the z/OS platform. The System SSL toolkit obtains its CA list from the underlying SSL repository (the SAF key ring or the HFS key database).

principal_sponsor

The `principal_sponsor` namespace stores configuration information to be used when obtaining credentials. `principal_sponsor` provides an implementation of a principal sponsor that creates credentials for applications automatically.

Use of the `PrincipalSponsor` is disabled by default and can only be enabled through configuration.

The `PrincipalSponsor` represents an entry point into the secure system. It must be activated and authenticate the user, before any application-specific logic executes. This allows unmodified, security-unaware applications to have `Credentials` established transparently, prior to making invocations.

In this section

The following variables are in this namespace:

- `use_principal_sponsor`
 - `auth_method_id`
 - `auth_method_data`
 - `callback_handler:ClassName`
 - `login_attempts`
-

use_principal_sponsor

`use_principal_sponsor` specifies whether an attempt is made to obtain credentials automatically. Defaults to `false`. If set to `true`, the following `principal_sponsor` variables must contain data in order for anything to actually happen.

auth_method_id

`auth_method_id` specifies the authentication method to be used. The following authentication methods are available:

<code>pkcs12_file</code>	The authentication method uses a PKCS#12 file. Not supported in z/OS.
<code>security_label</code>	z/OS only. The authentication data is specified by supplying the name of a label in a key ring.

For example, you can select the `pkcs12_file` authentication method as follows:

```
principal_sponsor:auth_method_id = "pkcs12_file";
```

auth_method_data

`auth_method_data` is a string array containing information to be interpreted by the authentication method represented by the `auth_method_id`.

For the `pkcs12_file` authentication method, the following authentication data can be provided in `auth_method_data`:

<code>filename</code>	A PKCS#12 file that contains a certificate chain and private key— <i>required</i> .
<code>password</code>	A password for the private key— <i>optional</i> . It is bad practice to supply the password from configuration for deployed systems. If the password is not supplied, the user is prompted for it.
<code>password_file</code>	The name of a file containing the password for the private key— <i>optional</i> . This option is not recommended for deployed systems.

For the `security_label` authentication method on z/OS, the following authentication data can be provided in `auth_method_data`:

<code>label</code>	The name of a label in a key ring.
--------------------	------------------------------------

For example, to configure an application on Windows to use a certificate, `bob.p12`, whose private key is encrypted with the `bobpass` password, set the `auth_method_data` as follows:

```
principal_sponsor:auth_method_data =  
  ["filename=c:\users\bob\bob.p12", "password=bobpass"];
```

The following points apply to Java implementations:

- If the file specified by `filename=` is not found, it is searched for on the classpath.
- The file specified by `filename=` can be supplied with a URL instead of an absolute file location.
- The mechanism for prompting for the password if the password is supplied through `password=` can be replaced with a custom mechanism, as demonstrated by the `login` demo.

- There are two extra configuration variables available as part of the `principal_sponsor` namespace, namely `principal_sponsor:callback_handler` and `principal_sponsor:login_attempts`. These are described below.
- These Java-specific features are available subject to change in future releases; any changes that can arise probably come from customer feedback on this area.

callback_handler:ClassName

`callback_handler:ClassName` specifies the class name of an interface that implements the interface `com.ionacorba.tls.auth.CallbackHandler`. This variable is only used for Java clients.

login_attempts

`login_attempts` specifies how many times a user is prompted for authentication data (usually a password). It applies for both internal and custom `CallbackHandlers`; if a `CallbackHandler` is supplied, it is invoked upon up to `login_attempts` times as long as the `PrincipalAuthenticator` returns `SecAuthFailure`. This variable is only used by Java clients.

principal_sponsor:csi

The `principal_sponsor:csi` namespace stores configuration information to be used when obtaining CSI (Common Secure Interoperability) credentials. It includes the following:

- `use_existing_credentials`
- `use_principal_sponsor`
- `auth_method_data`
- `auth_method_id`

use_existing_credentials

A boolean value that specifies whether ORBs that share credentials can also share CSI credentials. If `true`, any CSI credentials loaded by one credential-sharing ORB can be used by other credential-sharing ORBs loaded after it; if `false`, CSI credentials are not shared.

This variable has no effect, unless the `plugins:security:share_credentials_across_orbs` variable is also `true`. Default is `false`.

use_principal_sponsor

`use_principal_sponsor` is a boolean value that switches the CSI principal sponsor on or off.

If set to `true`, the CSI principal sponsor is enabled; if `false`, the CSI principal sponsor is disabled and the remaining `principal_sponsor:csi` variables are ignored. Defaults to `false`.

auth_method_data

`auth_method_data` is a string array containing information to be interpreted by the authentication method represented by the `auth_method_id`.

For the GSSUPMech authentication method, the following authentication data can be provided in `auth_method_data`:

<code>username</code>	The username for CSIV2 authorization. This is optional. Authentication of CSIV2 usernames and passwords is performed on the server side. The administration of usernames depends on the particular security mechanism that is plugged into the server side see auth_over_transport:authentication_service .
<code>password</code>	The password associated with username. This is optional. It is bad practice to supply the password from configuration for deployed systems. If the password is not supplied, the user is prompted for it.
<code>domain</code>	The CSIV2 authentication domain in which the username/password pair is authenticated. When the client is about to open a new connection, this domain name is compared with the domain name embedded in the relevant IOR (see policies:csi:auth_over_transport:server_domain_name). The domain names must match. Note: If <code>domain</code> is an empty string, it matches any target domain. That is, an empty domain string is equivalent to a wildcard.

If any of the preceding data are omitted, the user is prompted to enter authentication data when the application starts up.

For example, to log on to a CSIV2 application as the `administrator` user in the `US-SantaClara` domain:

```
principal_sponsor:csi:auth_method_data =
  ["username=administrator", "domain=US-SantaClara"];
```

When the application is started, the user is prompted for the administrator password.

Note: It is currently not possible to customize the login prompt associated with the CSIv2 principal sponsor. As an alternative, you could implement your own login GUI by programming and pass the user input directly to the principal authenticator.

auth_method_id

`auth_method_id` specifies a string that selects the authentication method to be used by the CSI application. The following authentication method is available:

GSSUPMech	The Generic Security Service Username/Password (GSSUP) mechanism.
-----------	---

For example, you can select the GSSUPMech authentication method as follows:

```
principal_sponsor:csi:auth_method_id = "GSSUPMech";
```

ASN.1 and Distinguished Names

The OSI Abstract Syntax Notation One (ASN.1) and X.500 Distinguished Names play an important role in the security standards that define X.509 certificates and LDAP directories.

In this appendix

This appendix contains the following section:

ASN.1	page 308
Distinguished Names	page 309

ASN.1

Overview

The *Abstract Syntax Notation One* (ASN.1) was defined by the OSI standards body in the early 1980s to provide a way of defining data types and structures that is independent of any particular machine hardware or programming language. In many ways, ASN.1 can be considered a forerunner of the OMG's IDL, because both languages are concerned with defining platform-independent data types.

ASN.1 is important, because it is widely used in the definition of standards (for example, SNMP, X.509, and LDAP). In particular, ASN.1 is ubiquitous in the field of security standards—the formal definitions of X.509 certificates and distinguished names are described using ASN.1 syntax. You do not require detailed knowledge of ASN.1 syntax to use these security standards, but you need to be aware that ASN.1 is used for the basic definitions of most security-related data types.

BER

The OSI's Basic Encoding Rules (BER) define how to translate an ASN.1 data type into a sequence of octets (binary representation). The role played by BER with respect to ASN.1 is, therefore, similar to the role played by GIOP with respect to the OMG IDL.

DER

The OSI's Distinguished Encoding Rules (DER) are a specialization of the BER. The DER consists of the BER plus some additional rules to ensure that the encoding is unique (BER encodings are not).

References

You can read more about ASN.1 in the following standards documents:

- ASN.1 is defined in X.208.
- BER is defined in X.209.

Distinguished Names

Overview

Historically, distinguished names (DN) were defined as the primary keys in an X.500 directory structure. In the meantime, however, DNs have come to be used in many other contexts as general purpose identifiers. In the Orbix Security Framework, DNs occur in the following contexts:

- X.509 certificates—for example, one of the DNs in a certificate identifies the owner of the certificate (the security principal).
- LDAP—DNs are used to locate objects in an LDAP directory tree.

String representation of DN

Although a DN is formally defined in ASN.1, there is also an LDAP standard that defines a UTF-8 string representation of a DN (see [RFC 2253](#)). The string representation provides a convenient basis for describing the structure of a DN.

Note: The string representation of a DN does *not* provide a unique representation of DER-encoded DN. Hence, a DN that is converted from string format back to DER format does not always recover the original DER encoding.

DN string example

The following string is a typical example of a DN:

```
C=US,O=Micro Focus,OU=Engineering,CN=A. N. Other
```

Structure of a DN string

A DN string is built up from the following basic elements:

- [OID](#).
- [Attribute types](#).
- [AVA](#).
- [RDN](#).

OID

An OBJECT IDENTIFIER (OID) is a sequence of bytes that uniquely identifies a grammatical construct in ASN.1.

Attribute types

The variety of attribute types that could appear in a DN is theoretically open-ended, but in practice only a small subset of attribute types are used. [Table 9](#) shows a selection of the attribute types that you are most likely to encounter:

Table 9: *Commonly Used Attribute Types*

String Representation	X.500 Attribute Type	Size of Data	Equivalent OID
C	countryName	2	2.5.4.6
O	organizationName	1...64	2.5.4.10
OU	organizationalUnitName	1...64	2.5.4.11
CN	commonName	1...64	2.5.4.3
ST	stateOrProvinceName	1...64	2.5.4.8
L	localityName	1...64	2.5.4.7
STREET	streetAddress		
DC	domainComponent		
UID	userid		

AVA

An *attribute value assertion* (AVA) assigns an attribute value to an attribute type. In the string representation, it has the following syntax:

```
<attr-type>=<attr-value>
```

For example:

```
CN=A. N. Other
```

Alternatively, you can use the equivalent OID to identify the attribute type in the string representation (see [Table 9](#)). For example:

```
2.5.4.3=A. N. Other
```

RDN

A *relative distinguished name* (RDN) represents a single node of a DN (the bit that appears between the commas in the string representation).

Technically, an RDN might contain more than one AVA (it is formally defined as a set of AVAs); in practice, however, this almost never occurs. In the string representation, an RDN has the following syntax:

`<attr-type>=<attr-value>[+<attr-type>=<attr-value> ...]`

Here is an example of a (very unlikely) multiple-value RDN:

`OU=Eng1+OU=Eng2+OU=Eng3`

Here is an example of a single-value RDN:

`OU=Engineering`

Association Options

This appendix describes the semantics of all the association options that are supported by Orbix.

In this appendix

This appendix contains the following section:

Association Option Semantics	page 314
--	--------------------------

Association Option Semantics

Overview

This appendix defines how `AssociationOptions` are used with `SecClientInvocation` and `SecTargetInvocation` policies.

IDL Definitions

`AssociationOptions` are enumerated in the CORBA security specification as follows:

```
//IDL
typedef unsigned short AssociationOptions;
const AssociationOptions NoProtection = 1;
const AssociationOptions Integrity = 2;
const AssociationOptions Confidentiality = 4;
const AssociationOptions DetectReplay = 8;
const AssociationOptions DetectMisordering = 16;
const AssociationOptions EstablishTrustInTarget = 32;
const AssociationOptions EstablishTrustInClient = 64;
// Unsupported option: NoDelegation
// Unsupported option: SimpleDelegation
// Unsupported option: CompositeDelegation
```

Table of association options

[Table 10](#) shows how the options affect client and target policies:

Table 10: *AssociationOptions for Client and Target*

Association Options	client_supports	client_requires	target_supports	target_requires
NoProtection	Client supports unprotected messages.	The client's minimal protection requirement is unprotected messages.	Target supports unprotected messages.	The target's minimal protection requirement is unprotected messages.
Integrity	The client supports integrity protected messages.	The client requires messages to be integrity protected.	The target supports integrity protected messages.	The target requires messages to be integrity protected.

Table 10: *AssociationOptions for Client and Target*

Association Options	client_supports	client_requires	target_supports	target_requires
Confidentiality	The client supports confidentiality protected messages.	The client requires messages to be confidentiality protected.	The target supports confidentiality protected messages.	The target requires messages to be confidentiality protected.
DetectReplay	The client can detect replay of requests (and request fragments).	The client requires detection of message replay.	The target can detect replay of requests (and request fragments).	The target requires detection of message replay.
DetectMisordering	The client can detect sequence errors of requests (and request fragments).	The client requires detection of message mis-sequencing.	The target can detect sequence errors of requests (and request fragments).	The target requires detection of message mis-sequencing.
EstablishTrustInTarget	The client is capable of authenticating the target.	The client requires establishment of trust in the target's identity.	The target is prepared to authenticate its identity to the client.	(This option is invalid).
EstablishTrustInClient	The client is prepared to authenticate its identity to the target.	(This option is invalid).	The target is capable of authenticating the client.	The target requires establishment of trust in the client's identity.

SSL/TLS Sample Configurations

This appendix provides some SSL/TLS sample configurations that you can use as a template for configuring your own applications.

In this appendix

This appendix contains the following section:

SSL/TLS Sample Configurations on z/OS

page 318

SSL/TLS Sample Configurations on z/OS

Overview

This section lists a variety of SSL/TLS sample configurations suitable for applications running on the z/OS platform. You can use these samples as a starting point for configuring your own applications.

For more details on SSL/TLS configuration, see [“Securing Communications with SSL/TLS” on page 36](#).

Client configurations

The following client configurations are included in [Example 20](#):

- `demos.tls.secure_client_with_cert`
 - `demos.tls.semi_secure_client_with_cert`
-

Server configurations

The following server configurations are included in [Example 20](#):

- `demos.tls.secure_server_no_client_auth`
 - `demos.tls.secure_server_enforce_client_auth`
 - `demos.tls.semi_secure_server_no_client_auth`
 - `demos.tls.semi_secure_server_enforce_client_auth`
-

Sample configurations

[Example 20](#) shows a variety of sample SSL/TLS configurations that you can copy or adapt for use in your own applications.

Example 20: SSL/TLS Configurations on the z/OS Platform

```
# Orbix Configuration File
demos
{
  ...
  tls
  {
    event_log:filters = ["IT_ATLI_TLS=*", "IT_IIOP=*",
"IT_IIOP_TLS=*", "IT_TLS=*", "IT_GenericSecurityToolkit=*"];

    policies:target_secure_invocation_policy:requires =
["Confidentiality", "EstablishTrustInClient"];
    policies:target_secure_invocation_policy:supports =
```

Example 20: SSL/TLS Configurations on the z/OS Platform

```

["Confidentiality", "Integrity", "DetectReplay",
 "DetectMisordering", "EstablishTrustInClient",
 "EstablishTrustInTarget"];

    policies:client_secure_invocation_policy:requires =
["Confidentiality", "EstablishTrustInTarget"];
    policies:client_secure_invocation_policy:supports =
["Confidentiality", "Integrity", "DetectReplay",
 "DetectMisordering", "EstablishTrustInClient",
 "EstablishTrustInTarget"];

    orb_plugins = ["local_log_stream", "iiop_profile",
"giop", "iiop_tls"];

    # tls demos use security labels to identify certificates
    # within keyrings
    # each keyring will be defined in subsequent scope
    principal_sponsor:use_principal_sponsor = "true";
    principal_sponsor:auth_method_id = "security_label";

secure_client_with_cert
{
    policies:client_secure_invocation_policy:requires =
["Confidentiality", "EstablishTrustInTarget"];
    policies:client_secure_invocation_policy:supports =
["Confidentiality", "Integrity", "DetectReplay",
 "DetectMisordering", "EstablishTrustInClient",
 "EstablishTrustInTarget"];

    principal_sponsor:use_principal_sponsor = "true";
    principal_sponsor:auth_method_data =
["label=bank_server"];

    plugins:systemssl_toolkit:saf_keyring = "ORBXRING";
};

semi_secure_client_with_cert
{
    orb_plugins = ["local_log_stream", "iiop_profile",
"giop", "iiop", "iiop_tls"];

    policies:client_secure_invocation_policy:requires =
["NoProtection"];

```

Example 20: SSL/TLS Configurations on the z/OS Platform

```

        policies:client_secure_invocation_policy:supports =
["NoProtection", "Confidentiality", "Integrity", "DetectReplay",
"DetectMisordering", "EstablishTrustInClient",
"EstablishTrustInTarget"];

        principal_sponsor:use_principal_sponsor = "true";
        principal_sponsor:auth_method_data =
["label=bank_server"];

        plugins:systemssl_toolkit:saf_keyring = "ORBXRING";
    };

    secure_server_no_client_auth
    {
        policies:target_secure_invocation_policy:requires =
["Confidentiality"];

        policies:target_secure_invocation_policy:supports =
["Confidentiality", "Integrity", "DetectReplay",
"DetectMisordering", "EstablishTrustInTarget"];

        principal_sponsor:use_principal_sponsor = "true";
        principal_sponsor:auth_method_data =
["label=bank_server"];

        plugins:systemssl_toolkit:saf_keyring = "ORBXRING";
    };

    secure_server_enforce_client_auth
    {
        policies:target_secure_invocation_policy:requires =
["EstablishTrustInClient", "Confidentiality"];
        policies:target_secure_invocation_policy:supports =
["EstablishTrustInClient", "Confidentiality", "Integrity",
"DetectReplay", "DetectMisordering",
"EstablishTrustInTarget"];

        principal_sponsor:use_principal_sponsor = "true";
        principal_sponsor:auth_method_data =
["label=bank_server"];

        plugins:systemssl_toolkit:saf_keyring = "ORBXRING";
    };

```

Example 20: SSL/TLS Configurations on the z/OS Platform

```

semi_secure_server_no_client_auth
{
    orb_plugins = ["local_log_stream", "iiop_profile",
"giop", "iiop", "iiop_tls"];

    policies:target_secure_invocation_policy:requires =
["NoProtection"];
    policies:target_secure_invocation_policy:supports =
["NoProtection", "Confidentiality", "Integrity", "DetectReplay",
"DetectMisordering", "EstablishTrustInTarget"];

    principal_sponsor:use_principal_sponsor = "true";
    principal_sponsor:auth_method_data =
["label=bank_server"];

    plugins:systemssl_toolkit:saf_keyring = "ORBXRING";
};

semi_secure_server_enforce_client_auth
{
    orb_plugins = ["iiop_profile", "giop", "iiop",
"iiop_tls", "local_log_stream"];

    policies:target_secure_invocation_policy:requires =
["NoProtection"];
    policies:target_secure_invocation_policy:supports =
["NoProtection", "Confidentiality", "Integrity", "DetectReplay",
"DetectMisordering", "EstablishTrustInClient",
"EstablishTrustInTarget"];

    principal_sponsor:use_principal_sponsor = "true";
    principal_sponsor:auth_method_data =
["label=bank_server"];

    plugins:systemssl_toolkit:saf_keyring = "ORBXRING";
};
};
...
};

```


Security Recommendations

This appendix lists some general recommendations for ensuring the effectiveness of Orbix security.

In this appendix

This appendix contains the following sections:

General Recommendations	page 324
Orbix Services	page 325

General Recommendations

List of recommendations

The following general recommendations can help you secure your system using Orbix applications

1. Use SSL security for every application wherever possible.
2. Use the strongest cipher suites available. There is little extra overhead if you use 128 bit instead of 40 bit encryption for a typical connection.
3. If your application must connect to insecure applications, limit the aspects of your system that use insecure communications to the minimum necessary using policies and security aware code.
4. Treat any IOR received from an insecure endpoint as untrustworthy. Set your policies so that you cannot use insecure IORs accidentally. Set all communications in your ORBs to be secure by default and use the appropriate policies to override these where necessary.
5. It is important to remember that the certificates supplied with Orbix are for demonstration purposes only and must be replaced with a securely generated set of real certificates before applications can run in a production environment.
6. The contents of your trusted CA list files must only include CA certificates that you trust.
7. Do not use passwords in the configuration file. This feature is only a developer aid.
8. The security of all SSL/TLS programs is only as strong as the weakest cipher suite that they support. Consider making stronger cipher suites available as an optional service which may be availed of by applications with stronger minimum security requirements.
The bad guys will of course choose to use the weakest cipher suites.
9. Depending on the sensitivity of your system an RSA key size greater than 512 bits might be appropriate. 1024 bit keys are significantly slower than 512 bit keys but are much more secure.

Orbix Services

No authorization support for Orbix services

The Orbix services—that is, the locator, the node daemon, the naming service, and the interface repository (IFR)—are not to be considered as fully secured in this release. While they can be configured to use SSL they do not apply any authorization to operations that clients perform. This still applies, to a lesser extent, even if the services are configured to only allow secure connections and to enforce client authentication, because all clients with trusted client certificates can modify the services at will. That is, the Orbix services provide no way to distinguish between ordinary users and users requiring administrative privileges (authorization is not supported by the services).

Action-Role Mapping DTD

This appendix presents the document type definition (DTD) for the action-role mapping XML file.

DTD file

The action-role mapping DTD is shown in [Example 21](#).

Example 21:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT action-name (#PCDATA)>
<!ELEMENT role-name (#PCDATA)>
<!ELEMENT server-name (#PCDATA)>
<!ELEMENT action-role-mapping (server-name, interface+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT interface (name, action-role+)>
<!ELEMENT action-role (action-name, role-name+)>
<!ELEMENT allow-unlisted-interfaces (#PCDATA)>
<!ELEMENT secure-system (allow-unlisted-interfaces*,
    action-role-mapping+)>
```

Action-role mapping elements

The elements of the action-role mapping DTD can be described as follows:

```
<!ELEMENT action-name (#PCDATA)>
```

Specifies the action name to which permissions are assigned. The interpretation of the action name depends on the type of application:

- ◆ CORBA server—for IDL operations, the action name corresponds to the GIOP on-the-wire format of the operation name (usually the same as it appears in IDL).

For IDL attributes, the accessor or modifier action name corresponds to the GIOP on-the-wire format of the attribute accessor or modifier. For example, an IDL attribute, `foo`, would have an accessor, `_get_foo`, and a modifier, `_set_foo`.

- ◆ Artix server—for WSDL operations, the action name is equivalent to a WSDL operation name; that is, the `OperationName` from a tag, `<operation name="OperationName">`.

`<!ELEMENT action-role (action-name, role-name+)>`

Groups together a particular action and all of the roles permitted to perform that action.

`<!ELEMENT action-role-mapping (server-name, interface+)>`

Contains all of the permissions that apply to a particular server application.

`<!ELEMENT allow-unlisted-interfaces (#PCDATA)>`

Specifies the default access permissions that apply to interfaces not explicitly listed in the action-role mapping file. The element contents can have the following values:

- ◆ `true`—for any interfaces not listed, access to all of the interfaces' actions is allowed for all roles. If the remote user is unauthenticated (in the sense that no credentials are sent by the client), access is also allowed.

Note: However, if `<allow-unlisted-interfaces>` is `true` and a particular interface is listed, then only the actions explicitly listed within that interface's `interface` element are accessible. Unlisted actions from the listed interface are not accessible.

- ◆ `false`—for any interfaces not listed, access to all of the interfaces' actions is denied for all roles. Unauthenticated users are also denied access.

Default is `false`.

<!ELEMENT interface (name, action-role+)>

In the case of a CORBA server, the `interface` element contains all of the access permissions for one particular IDL interface.

In the case of an Artix server, the `interface` element contains all of the access permissions for one particular WSDL port type.

<!ELEMENT name (#PCDATA)>

Within the scope of an `interface` element, identifies the interface (IDL interface or WSDL port type) with which permissions are being associated. The format of the interface name depends on the type of application, as follows:

- ♦ CORBA server—the `name` element identifies the IDL interface using the interface's OMG repository ID. The repository ID normally consists of the characters `IDL:` followed by the fully scoped name of the interface (using `/` instead of `::` as the scoping character), followed by the characters `:1.0`. Hence, the `Simple::SimpleObject` IDL interface is identified by the `IDL:Simple/SimpleObject:1.0` repository ID.

Note: The form of the repository ID can also be affected by various `#pragma` directives appearing in the IDL file. A commonly used directive is `#pragma prefix`.

For example, the `CosNaming::NamingContext` interface in the naming service module, which uses the `omg.org` prefix, has the following repository ID: `IDL:omg.org/CosNaming/NamingContext:1.0`

- ♦ Artix server—the `name` element contains a WSDL port type name, specified in the following format:

NamespaceURI:PortTypeName

The *PortTypeName* comes from a tag, `<portType name="PortTypeName">`, defined in the *NamespaceURI* namespace.

The *NamespaceURI* is usually defined in the `<definitions targetNamespace="NamespaceURI" ...>` tag of the WSDL contract.

<!ELEMENT role-name (#PCDATA)>

Specifies a role to which permission is granted. The role name can be any role that belongs to the server's Artix authorization realm (for CORBA bindings, the realm name is specified by the

`plugins:gsp:authorization_realm` configuration variable; for SOAP bindings, the realm name is specified by the `plugins:asp:authorization_realm` configuration variable) or to the `IONAGlobalRealm` realm. The roles themselves are defined in the security server backend; for example, in a file adapter file or in an LDAP backend.

```
<!ELEMENT secure-system (allow-unlisted-interfaces*,
action-role-mapping+)>
```

The outermost scope of an action-role mapping file groups together a collection of `action-role-mapping` elements.

```
<!ELEMENT server-name (#PCDATA)>
```

The `server-name` element specifies the configuration scope (that is, the ORB name) used by the server in question. This is normally the value of the `-ORBname` parameter passed to the server executable on the command line.

Index

Symbols

- #pragma prefix 66
- <action-role-mapping> tag 66, 74
- <allow-unlisted-interfaces> tag 65
- <interface> tag 66
- <name> tag 66
- <server-name> tag 66, 72

A

- AccessId credentials attribute 222
- AccessId security attribute 222
- ACL
 - <action-role-mapping> tag 66
 - <allow-unlisted-interfaces> tag 65
 - <interface> tag 66
 - <name> tag 66
 - <server-name> tag 66
 - action_role_mapping configuration variable 64
 - action_role_mapping file 64
 - action-role mapping file, example 65
 - centralized 68, 71
 - ClientAccessDecision interface 70, 72, 77
 - com.iona.isp.authz.adapters property 73
 - localized 69
 - plugins:gsp:acl_policy_data_id variable 75, 76
 - plugins:gsp:action_role_mapping_file variable 69
 - plugins:gsp:authorization_policy_enforcement_point variable 73
 - plugins:gsp:authorization_policy_store_type variable 73
- action_role_mapping configuration variable 64
- action-role mapping file
 - <action-role-mapping> tag 66
 - <allow-unlisted-interfaces> tag 65
 - <interface> tag 66
 - <name> tag 66
 - <server-name> tag 66
- CORBA
 - configuring 64
 - example 65
- application-level security 150
- ASN.1 80
- association options
 - and mechanism policy 114
 - client secure invocation policy, default 110
 - DetectMisordering 205
 - DetectReply 205
 - EstablishTrustInClient 42, 57, 137
 - EstablishTrustInClient, CSiv2 164, 165
 - EstablishTrustInTarget 134, 137
 - IdentityAssertion, CSiv2 185
 - NoProtection 45
 - rules of thumb 114
 - SSL/TLS
 - Confidentiality 108
 - DetectMisordering 108
 - DetectReplay 108
 - EstablishTrustInClient 109
 - EstablishTrustInTarget 109
 - Integrity 108
 - NoProtection 108
 - setting 106
 - target secure invocation policy, default 112
- Asymmetric cryptography 25
- AttributeList type 221
- attribute service policy 184
- AttributeService policy data 207
- Attribute value assertions, See AVA
- authenticate() operation 211, 212
- Authentication 22, 24
- authentication
 - and mechanism policy 137
 - caching of credentials 60
 - CSiv2, client configuration 173
 - CSiv2, requiring 164
 - CSiv2, sample configuration 172
 - CSiv2, server configuration 175
 - CSiv2 client-side policy 206
 - CSiv2 server-side policy 206
 - EstablishTrustPolicy 203
 - GSSUP mechanism
 - invocation credentials 204
 - iSF
 - process of 49
 - IT_CSI_AUTH_METH_USERNAME_PASSWORD authentication method 217

- IT_TLS_AUTH_METH_CERT_CHAIN
 - authentication method 214
- IT_TLS_AUTH_METH_CERT_CHAIN_FILE
 - authentication method 214
- IT_TLS_AUTH_METH_LABEL authentication
 - method 214
- IT_TLS_AUTH_METH_PKCS11 authentication
 - method 214
- IT_TLS_AUTH_METH_PKCS12_DER
 - authentication method 214
- methods for SSL/TLS 214
- multiple own certificates 140
- over transport, in CSv2 158
- own certificate, specifying 139
- principal authenticator 210
- security capsule 211
- SSL/TLS
 - principal sponsor 140
 - requiring 132
 - target and client 136
 - target only 133
- authentication_cache_size configuration variable 60
- authentication_cache_timeout configuration
 - variable 60
- authentication over transport
 - client authentication token 160
 - client support, enabling 164
 - dependency on SSL/TLS 158
 - description 150, 158
 - own credentials 212
 - scenario 153
 - server configuration 165
 - SSL/TLS prerequisites 162
 - target requirements 165
 - target support, enabling 165
- authentication service
 - sample implementation 167
- authentication service class
 - specifying 166
- authentication service object
 - and CSI_SERVER_AS_POLICY policy 167
 - default implementation 167
 - iSF implementation 167
 - registering as an initial reference 167
- AuthenticationService policy data 206, 207
- AuthenticationService policy value 167
- authorization
 - caching of credentials 60
- automatic connection management

- interaction with rebind policy 227
- AVA
 - in distinguished names 238
- AVAList interface 238

B

- backward trust 57, 183
- Baltimore toolkit
 - selecting for C++ applications 255

C

- CA 27, 80
 - adding to a user key ring 92
 - choosing a host 84
 - commercial CAs 83
 - importing into RACF 91
 - in PKCS#12 file 139
 - list of trusted 86
 - multiple CAs 86
 - private CAs 84
 - security precautions 84
 - See *Alsocertificate* authority
- caching
 - authentication_cache_size configuration
 - variable 60
 - authentication_cache_timeout configuration
 - variable 60
 - of credentials 60
- centralized ACL 73
 - <action-role-mapping> tag 74
 - <server-name> tag 72
 - ClientAccessDecision interface 72
 - com.iona.isp.authz.adapters property 73
 - file list 74
 - is2.properties file 73
 - overview 68, 71
 - plugins:gsp:acl_policy_data_id variable 75, 76
 - plugins:gsp:authorization_policy_enforcement_point variable 73
 - plugins:gsp:authorization_policy_store_type
 - variable 73
 - selecting an ACL file 74
 - selection by ACL key 76
 - selection by ORB name 74
 - selection by override value 75
- CERTAUTH 91
- CertConstraintsPolicy 144, 253
- CertConstraintsPolicy policy 144, 235, 239, 253

- CertConstraints string array 241
- certificate authority
 - and certificate signing 80
- certificate constraints policy 235
 - C++ example 241
 - configuration, setting by 240
 - identity assertion and 183
 - Java example 243
 - programming, setting by 240
 - setting 240
 - three-tier target server 57
- certificate_constraints_policy variable 144, 253
- Certificates 25, 27
 - chain length 143
 - constraints 144, 253
 - contents of 237
 - validating 233–237
 - validation process 234
- certificates
 - adding to a user key ring 92
 - C++ parsing
 - get_issuer_dn_string() operation 238
 - get_subject_dn_string() operation 238
 - CertConstraintsPolicy policy 144, 253
 - chaining 85
 - common names 237
 - constraint language 144, 253
 - constraint policy, C++ example 241
 - constraint policy, Java example 243
 - constraints, applying 240
 - constraints policy 57
 - contents 237
 - contents of 80
 - creating using RACF 94
 - default validation 235
 - DER format 247
 - FTP transfer to OS/390 90
 - importing into RACF 89, 91
 - importing the CA into RACF 91
 - issuer 237
 - length limit 86
 - MaxChainLengthPolicy 143
 - multiple own certificates 140
 - obtaining 247
 - on OS/390 88
 - own, specifying 139
 - parsing
 - AVAList interface 238
 - peer 85
 - PKCS#12 file 87, 139
 - public key 81, 237
 - public key encryption 121
 - security handshake 133, 137
 - self-signed 85
 - serial number 81, 237
 - signing 80
 - specifying a source in OS/390 95
 - subject name 237
 - syntax 237
 - validation
 - validate_cert_chain() operation 244
 - validation, implementing 243
 - viewing in RACF 92
 - X.509 80
 - X.509 extensions 238
 - X509CertificateFactory interface 247
 - certificate validation
 - CertValidator interface 235
 - custom 235
 - default validation 235
 - certificate validation policy 234
 - implementing 243
 - CertificateValidatorPolicy policy 239
 - certification authority
 - on OS/390 89
 - Certification Authority. See CA
 - CertValidator interface 235
 - implementing 243
 - CertValidatorPolicy policy 235
 - chaining of certificates 85
 - Ciphersuites
 - choosing 324
 - cipher suites
 - ciphersuites configuration variable 127
 - default list 129
 - definitions 122
 - encryption algorithm 121
 - exportable 122
 - key exchange algorithm 121
 - mechanism policy 126
 - order of 128
 - secure hash algorithm 121
 - secure hash algorithms 122
 - security algorithms 121
 - specifying 119
 - ciphersuites configuration variable 127
 - ClientAccessDecision interface 70, 72, 77
 - client authentication token

- CSlv2 authentication over transport 160
- client_binding_list configuration variable 184
 - and CSlv2 authentication 164
 - iSF, client configuration 50
 - secure client 41
- client secure invocation policy
 - HTTPS 110
 - IIOp/TLS 110
- ClientSecureInvocationPolicy policy 107
- client-side policies 196
- client_version_policy
 - IIOp 291
- clustering, and fixed ports 47
- colocated invocations
 - and secure associations 104
- com.iona.isp.authz.adapters property 73
- common names 237
- common secure interoperability, see CSlv2
- Confidentiality association option 108
 - hints 116
- Confidentiality option 108
- connection_attempts 291
- constraint language 144, 253
- Constraints
 - for certificates 144, 253
- Contents of certificates 237
- CORBA
 - ACLs 62
 - action-role mapping file 64
 - action-role mapping file, example 65
 - intermediate server configuration 54
 - iSF, three-tier system 53
 - security, overview 34
 - SSL/TLS
 - client configuration 40
 - securing communications 36
 - server configuration 42
 - three-tier target server configuration 56
 - two-tier systems 48
- CORBA policies
 - how to set 196
- CORBA security
 - CSlv2 plug-in 35
 - IIOp/TLS plug-in 35
- CORBA Security RTF 1.7 22
- create_POA() operation
 - and policies 196
- create_policy() operation 242
- Credentials
 - retrieving 222
- credentials
 - AccessId attribute 222
 - AttributeList type 221
 - attributes, Orbix-specific 222
 - creating CSlv2 credentials 217
 - creating own 211
 - definition 221
 - get_target_credentials() operation 222
 - invocation credentials 204
 - obtaining 221
 - own
 - C++ example 225
 - parsing 225
 - own, creating multiple 212
 - own, CSlv2 212
 - own, SSL/TLS 212
 - _Public attribute 222
 - received 222
 - C++ example 231
 - received, SSL/TLS
 - parsing 232
 - retrieving 222
 - retrieving own 223
 - C++ example 224
 - retrieving received 230
 - retrieving target 226
 - SecurityAttributeType type 221
 - sharing 141, 168, 212
 - target, interaction with rebind policy 227
 - target, retrieving
 - C++ example 228
 - target, SSL/TLS
 - C++ example 229
 - parsing 229
- Credentials interface 211, 221
 - get_attributes() operation 221
 - Orbix-specific 222
- Cryptography
 - asymmetric 25
 - RSA. See RSA cryptography
 - symmetric 25, 28
- CSI_CLIENT_AS_POLICY policy type 206
- CSI_CLIENT_SAS_POLICY policy type 207
- CSICredentials interface 213
- CSI interceptor 50
- CSI plug-in
 - and CSlv2 principal sponsor 168
 - loading for Java applications 164

- csi plug-in 184
 - CSI_SERVER_AS_POLICY policy 167
 - CSI_SERVER_AS_POLICY policy type 207
 - CSI_SERVER_SAS_POLICY policy type 207
 - CSlv2
 - applicability 151
 - application-level security 150
 - association options 165
 - IdentityAssertion 185
 - attribute service policy 184
 - authentication, client configuration 173
 - authentication, Java example 217
 - authentication, requiring 164
 - authentication, sample configuration 172
 - authentication, server configuration 175
 - authentication over transport 150
 - authentication over transport, description 158
 - authentication over transport, own credentials 212
 - authentication over transport scenario 153
 - authentication policy, client-side 206
 - authentication policy, server-side 206
 - authentication scenario 158
 - authentication service 166
 - authentication service object 161
 - backward trust 183
 - certificate constraints policy 57, 183
 - client authentication token 160
 - client_binding_list configuration variable 184
 - csi plug-in for Java applications 184
 - features 150
 - GSSUPAuthData interface 217
 - GSSUP mechanism 158
 - identity assertion 151
 - own credentials 213
 - identity assertion, description 178
 - identity assertion, enabling 184
 - identity assertion, scenario description 179
 - identity assertion scenario 154
 - identity token types 181
 - intermediate server 154
 - ITTAbsent identity token type 181
 - ITTAnonymous identity token type 181
 - ITTPrincipalName identity token type 181
 - level 0 158
 - login 153
 - login, by configuration 170
 - login, by programming 170
 - login, dialog prompt 169
 - login options 169
 - policies 206
 - principal sponsor
 - client configuration 51
 - principal sponsor, description 168
 - principal sponsor, disabling 170
 - principal sponsor, enabling 168
 - principal_sponsor:csi:auth_method_data configuration variable 170
 - principal sponsor and client authentication token 161
 - received credentials 181
 - sample configurations 186
 - scenarios 152
 - server_binding_list configuration variable 184
 - SSL/TLS mutual authentication 182
 - SSL/TLS prerequisites 162, 182
 - SSL/TLS principal sponsor 183
 - transmitting security data 150
 - username and password, providing 168
 - CSlv2 authentication domain
 - and server domain name 165
 - CSlv2 plug-in
 - CORBA security 35
 - CSv2
 - CSICredentials interface 213
 - Current interface
 - and credentials 222
 - retrieving received credentials 231
 - custom validation 235
- D**
- data encryption standard
 - see DES
 - datasets
 - on OS/390 89
 - delegation
 - and identity assertion 178
 - DER format 247
 - DES
 - symmetric encryption 121
 - DetectMisordering association option 108, 205
 - hints 116
 - DetectMisordering option 108
 - DetectReplay association option 108
 - hints 116
 - DetectReplay option 108
 - DetectReply association option 205
 - DIRECT_PERSISTENCE policy value 46

- distinguished names 238
- domain name
 - and CSIV2 authentication over transport 150
 - ignored by iSF 49
- domain names
 - server domain name 165

E

- Encryption 22
- encryption algorithm
 - RC4 121
- encryption algorithms 121
 - DES 121
 - symmetric 121
 - triple DES 121
- EstablishTrustInClient
 - CSIV2 association option 164, 165, 169
- EstablishTrustInClient association option 42, 109, 137
 - hints 115
 - three-tier target server 57
- EstablishTrustInClient option 109
- EstablishTrustInTarget association option 109, 134, 137
 - hints 115
- EstablishTrustInTarget option 109
- EstablishTrustPolicy policy 203
 - and interaction between policies 205
- EstablishTrust type 203
- exportable cipher suites 122
- Extension interface 238
- ExtensionList interface 238

F

- fixed ports 46
 - DIRECT_PERSISTENCE policy value 46
 - host 47
 - IIOPTLS addr_list 47
 - IIOPTLS listen_addr 47
 - IIOPTLS port 47
 - INDIRECT_PERSISTENCE policy value 46

G

- generic security service username/password
 - mechanism
- get_attributes() operation
 - in Credentials interface 221
- get_issuer_dn_string() operation 238

- get_subject_dn_string() operation 238
- get_target_credentials() operation 222
- GIOP
 - and CSIV2 150
- GSP plug-in
 - and ClientAccessDecision 70
 - authentication_cache_size configuration variable 60
 - authentication_cache_timeout configuration variable 60
 - caching of credentials 60
- GSSUPAuthData interface 217
- GSSUPAuthData struct 220
- GSSUP mechanism 158
 - and CSIV2 principal sponsor 168
- GSSUP username 181

H

- Handshake, TLS 25
- HFS key database
 - and Orbix configuration 95
 - setting up 95
- HFS key databases 88
- hfs_keyring_filename configuration variable 95
- hfs_keyring_file_password configuration variable 95
- hfs_keyring_file_stashfile configuration variable 95
- HTTPS
 - ciphersuites configuration variable 127

I

- identity assertion
 - backward trust 183
 - certificate constraints policy 183
 - csi plug-in for Java applications 184
 - description 151, 178
 - enabling 184
 - intermediate server configuration 184
 - own credentials 213
 - policy, client-side 207
 - policy, server-side 207
 - received credentials and 181
 - sample client configuration 187
 - sample configurations 186
 - sample intermediate server configuration 189
 - sample target server configuration 191
 - scenarioCSIV2
 - identity assertion scenario 178
 - scenario description 179

- SSL/TLS dependency 178
 - SSL/TLS mutual authentication 182
 - SSL/TLS prerequisites 182
 - SSL/TLS principal sponsor 183
 - IdentityAssertion CSiv2 association option 185
 - identity assertion scenario 154
 - identity tokens
 - GSSUP username 181
 - subject DN in 181
 - types of 181
 - IIOp
 - and CSiv2 150
 - IIOp/TLS
 - ciphersuites configuration variable 127
 - host 47
 - IIOp/TLS addr_list 47
 - IIOp/TLS listen_addr 47
 - IIOp/TLS plug-in
 - CORBA security 35
 - IIOp/TLS port 47
 - IIOp plug-in
 - and semi-secure clients 41
 - IIOp policies 282, 289
 - client version 291
 - connection attempts 291
 - export hostnames 297
 - export IP addresses 297
 - GIOP version in profiles 297
 - server hostname 296
 - TCP options
 - delay connections 298
 - receive buffer size 298
 - IIOp policy
 - ports 296
 - IIOp_TLS interceptor 41
 - impersonation
 - and identity assertion 178
 - INDIRECT_PERSISTENCE policy value 46
 - initial references
 - IT_CSIAuthenticationObject 167
 - insecure object references
 - and QOP policy 202
 - Integrity 24, 29
 - Integrity association option 108
 - hints 116
 - Integrity option 108
 - intermediate server
 - and CSiv2 identity assertion 154
 - SSL/TLS connection from 180
 - intermediate server configuration 184
 - International Telecommunications Union 27
 - interoperability
 - z/OS, SSL/TLS 126
 - InvocationCredentialsPolicy policy 204
 - invocation policies
 - interaction with mechanism policy 114
 - is2.properties file 73
 - iSF
 - authentication service implementation 167
 - client configuration
 - CSI interceptor 50
 - CORBA
 - three-tier system 53
 - three-tier target server configuration 56
 - two-tier scenario description 49
 - CORBA security 34
 - domain name, ignoring 49
 - intermediate server configuration 54
 - server configuration
 - server_binding_list 50
 - server domain name, ignored 165
 - server_domain_name configuration variable 52
 - three-tier scenario description 54
 - two-tier CORBA systems 48
 - IT_CSIAuthenticationObject initial object ID 167
 - IT_CSI_AUTH_METH_USERNAME_PASSWORD
 - authentication method 217
 - ITTAbsent identity token type 181
 - ITTAnonymous identity token type 181
 - IT_TLS_AUTH_METH_CERT_CHAIN authentication
 - method 214
 - IT_TLS_AUTH_METH_CERT_CHAIN_FILE
 - authentication method 214
 - IT_TLS_AUTH_METH_LABEL authentication
 - method 214
 - IT_TLS_AUTH_METH_PKCS11 authentication
 - method 214
 - IT_TLS_AUTH_METH_PKCS12_DER authentication
 - method 214
 - ITTPrincipalName identity token type 181
 - ITU 27
- ## J
- JCE architecture
 - enabling 256

K

key database
 on OS/390 88
 key exchange algorithms 121

L

LifespanPolicy policy 46
 local ACL 69
 local_hostname 296
 localized ACL
 ClientAccessDecision interface 77
 logging
 in secure client 41
 login
 CSiv2 153
 CSiv2, by configuration 170
 CSiv2, by programming 170
 CSiv2 dialog prompt 169
 CSiv2 options 169

M

MAC 29
 max_chain_length_policy configuration variable 143
 MaxChainLengthPolicy policy 143
 MD5 122
 MechanismPolicy 108
 mechanism policy 126
 and authentication 137
 and interaction between policies 205
 interaction with invocation policies 114
 MechanismPolicy policy
 and interaction between policies 205
 message authentication code 29
 message digest 5
 see MD5
 message digests 108
 message fragments 108
 Message integrity 22
 minimum security levels 200
 mixed configurations, SSL/TLS 44
 multi-homed hosts, configure support for 296
 multiple CAs 86
 multiple own certificates 140
 mutual authentication
 identity assertion scenario 182

N

names, distinguished 238

namespace
 plugins:csi 257
 plugins:gsp 258
 policies 272
 policies:csi 279
 policies:https 282
 policies:iioptls 288
 principal_sponsor:csi 304
 principle_sponsor 300
 no_delay 298
 NoProtection association option 45, 108
 hints 117
 rules of thumb 114
 semi-secure applications 117
 NoProtection option 108

O

object-level policies
 invocation credentials policy 204
 object references
 and target credentials 227
 making insecure 202
 ORB
 security capsule 211
 orb_plugins configuration variable 41
 client configuration 50
 orb_plugins list
 CSI plug-in, including the 164
 orb_plugins variable
 and the NoProtection association option 117
 semi-secure configuration 118
 OS/390
 FTP transfer of certificates 90
 OS/390 certificate management 88
 own credentials
 creating 211
 creating multiple 212
 CSICredentials interface 213
 CSiv2 212
 definition 221
 principal authenticator 211
 retrieving 223
 C++ example 224
 SSL/TLS 212
 C++ example 225
 parsing 225
 TLSCredentials interface 212

P

- passwords
 - for RACF certificates 92
 - hfs_keyring_file_password 95
 - hfs_keyring_file_stashfile 95
- PDK
 - and custom SSL/TLS toolkit 100
- peer certificate 85
- performance
 - caching of credentials 60
- PersistenceModePolicy policy 46
- PKCS#12 certificates
 - viewing in RACF 92
- PKCS#12 files 139
 - definition 87
 - private key 139
- plug-in development kit 100
- plug-ins
 - csi 184
 - CSI, and CSiv2 principal sponsor 168
 - CSiv2, in CORBA security 35
 - IIOp 41
 - IIOp/TLS, in CORBA security 35
- plugins:csi:shlib_name 257
- plugins:csi:use_legacy_policies 257
- plugins:gsp:acl_policy_data_id variable 75, 76
- plugins:gsp:action_role_mapping_file variable 69, 73
- plugins:gsp:authorization_policy_enforcement_point variable 73
- plugins:gsp:authorization_policy_store_type variable 73
- plugins:gsp:authorization_realm 259
- plugins:iioptcp_listener:reincarnate_attempts 268
- plugins:iioptcp_listener:reincarnation_retry_backoff_ratio 268
- plugins:iioptcp_listener:reincarnation_retry_delay 268
- plugins:iioptls:hfs_keyring_filename 270
- plugins:iioptls:hfs_keyring_file_password 292
- plugins:iioptls:hfs_keyring_file_stashfile 270
- plugins:iioptls:racf_keyring 271
- plugins:iioptls:tcp_listener:reincarnation_retry_backoff_ratio 268
- plugins:iioptls:tcp_listener:reincarnation_retry_delay 268
- polices:max_chain_length_policy 274
- policies
 - and create_POA() operation 196
 - and_set_policy_overrides() operation 196
 - C++ example 197
 - CertConstraintsPolicy 144, 239, 253
 - certificate constraints 235, 240
 - certificate validation 234
 - CertificateValidatorPolicy 239
 - ClientSecureInvocationPolicy 107
 - client-side 196
 - CSI_SERVER_AS_POLICY 167
 - CSiv2, programmable 206
 - EstablishTrustPolicy 203
 - how to set 196
 - HTTPS
 - client secure invocation 110
 - target secure invocation 112
 - identity assertion, client-side 207
 - identity assertion, server-side 207
 - IIOp/TLS
 - client secure invocation 110
 - target secure invocation 112
 - insecure object references 202
 - interaction between 205
 - InvocationCredentialsPolicy policy 204
 - MaxChainLengthPolicy 143
 - minimum security levels 200
 - PolicyCurrent type 196
 - PolicyManager type 196
 - QOPP policy 202
 - rebind policy 227
 - restricting cipher suites 202
 - SecClientSecureInvocation 110
 - SecClientSecureInvocation policy 200
 - SecQOPConfidentiality enumeration value 202
 - SecQOPIntegrityAndConfidentiality enumeration value 202
 - SecQOPIntegrity enumeration value 202
 - SecQOPNoProtection enumeration value 202
 - SecTargetSecureInvocation 112
 - SecTargetSecureInvocation policy 200
 - server-side 196
 - SSL/TLS 199
 - TargetSecureInvocationPolicy 107
 - TLS_CERT_CONSTRAINTS_POLICY 242
- policies:allow_unauthenticated_clients_policy 272
- policies:certificate_constraints_policy 273
- policies:csi:attribute_service:client_supports 279
- policies:csi:attribute_service:target_supports 280
- policies:csi:auth_over_transport:target_supports 281
- policies:csi:auth_over_transport:authentication_servi

- ce configuration variable 166, 167
- policies:csi:auth_over_transport:client_supports 280
- policies:csi:auth_over_transport:client_supports configuration variable 164
- policies:csi:auth_over_transport:target_requires 281
- policies:csi:auth_over_transport:target_requires configuration variable 165
- policies:csi:auth_over_transport:target_supports configuration variable 165
- policies:https:allow_unauthenticated_clients_policy 282
- policies:https:certificate_constraints_policy 283
- policies:https:client_secure_invocation_policy:requires 283
- policies:https:client_secure_invocation_policy:supports 283
- policies:https:max_chain_length_policy 284
- policies:https:mechanism_policy:ciphersuites 284
- policies:https:mechanism_policy:protocol_version 286
- policies:https:session_caching_policy 287
- policies:https:target_secure_invocation_policy:requires 287
- policies:https:target_secure_invocation_policy:supports 287
- policies:https:trusted_ca_list_policy 288
- policies:iioptls:allow_unauthenticated_clients_policy 290
- policies:iioptls:certificate_constraints_policy 290
- policies:iioptls:client_secure_invocation_policy:requires 290
- policies:iioptls:client_secure_invocation_policy:supports 291
- policies:iioptls:client_version_policy 291
- policies:iioptls:connection_attempts 291
- policies:iioptls:connection_retry_delay 291
- policies:iioptls:max_chain_length_policy 292
- policies:iioptls:mechanism_policy:ciphersuites 293
- policies:iioptls:mechanism_policy:protocol_version 295
- policies:iioptls:server_address_mode_policy:local_hostname 296
- policies:iioptls:server_address_mode_policy:port_range 296
- policies:iioptls:server_address_mode_policy:publish_hostname 297
- policies:iioptls:server_version_policy 297
- policies:iioptls:target_secure_invocation_policy:requires 297
- policies:iioptls:target_secure_invocation_policy:supports 298
- policies:iioptls:tcp_options:send_buffer_size 299
- policies:iioptls:tcp_options_policy:no_delay 298
- policies:iioptls:tcp_options_policy:recv_buffer_size 298
- policies:iioptls:trusted_ca_list_policy 299
- policies:mechanism_policy:ciphersuites 274
- policies:mechanism_policy:protocol_version 276
- policies:target_secure_invocation_policy:requires 277
- policies:target_secure_invocation_policy:supports 277
- policies:trusted_ca_list_policy 278
- PolicyCurrent type 196
- policy data
 - AttributeService 207
 - AuthenticationService 206, 207
- PolicyList interface 241
- PolicyList object 197
- PolicyManager interface 241
- PolicyManager object 197
- PolicyManager type 196
- policy types
 - CSI_CLIENT_AS_POLICY 206
 - CSI_CLIENT_SA_S_POLICY 207
 - CSI_SERVER_AS_POLICY 207
 - CSI_SERVER_SA_S_POLICY 207
- policy values
 - AuthenticationService 167
- principal
 - definition 211
- principal authenticator
 - authenticate() operation 211, 212
 - CSlv2
 - Java example 217
 - definition 211
 - security capsule 211
 - SSL/TLS
 - C++ example 214
 - using 210
- principal sponsor
 - CSlv2
 - client configuration 51
 - CSlv2, description 168
 - CSlv2 and client authentication token 161
 - SSL/TLS
 - definition 140

- enabling 43
- SSL/TLS, disabling 41
- principal_sponsor:csi:auth_method_data 305
- principal_sponsor:csi:auth_method_data
 - configuration variable 169, 170
- principal_sponsor:csi:use_method_id configuration
 - variable 168
- principal_sponsor:csi:use_principal_sponsor 304
- principal_sponsor:csi:use_principal_sponsor
 - configuration variable 168, 170
- principal_sponsor Namespace Variables 300
- principal_sponsors
 - CSlv2, disabling 170
 - CSlv2, enabling 168
 - SSL/TLS, and CSlv2 163
- PrincipleAuthenticator interface 212, 215, 220
- principle_sponsor:auth_method_data 301
- principle_sponsor:auth_method_id 301
- principle_sponsor:callback_handler:ClassName 303
- principle_sponsor:login_attempts 303
- principle_sponsor:use_principle_sponsor 300
- Privacy 24
- private key
 - in PKCS#12 file 139
- Protocol, TLS handshake 25
- protocol version
 - interoperability with z/OS 126
- protocol_version configuration variable 126
- _Public credentials attribute 222
- public key 237
- Public key cryptography 25
- public key encryption 121
- public keys 81
- _Public security attribute 222
- publish_hostname 297

Q

- QOP enumerated type 202
- QOP policy
 - restricting cipher suites 202
- QOPPolicy policy 202
 - and interaction between policies 205
- quality of protection 202

R

- RACDCERT command 89, 91
- RACF 88
 - allocating datasets 89

- creating certificates 94
- importing certificates 89, 91
- RACF certificates
 - password 92
- RACF key ring
 - and Orbix configuration 95
- racf_keyring configuration variable 95
- RC4 encryption 121
- rebind policy
 - interaction with target credentials 227
- received credentials
 - Current object 231
 - definition 221
 - identity assertion and 181
 - retrieving 230
 - C++ example 231
 - SSL/TLS
 - parsing 232
- ReceivedCredentials interface 155, 221
 - Orbix-specific 222
 - parsing received credentials 232
- recv_buffer_size 298
- remote method invocation, see RMI
- Replay detection 108
- repository ID
 - #pragma prefix 66
 - in action-role mapping file 66
- required security features 201
- Rivest Shamir Adleman
 - see RSA
- Rivest Shamir Adleman cryptography. See RSA
 - cryptography
- RMI/IIOP
 - and CSlv2 150
- root certificate directory 86
- RSA 121
 - key size 324
 - symmetric encryption algorithm 121
- RSA cryptography 25

S

- scenarios
 - authentication in CSlv2 158
 - authentication over transport 153
 - CSlv2 152
 - identity assertion 154
- Schannel toolkit
 - selecting for C++ applications 255
- SecClientSecureInvocation policy 110, 200

- SecQOPConfidentiality enumeration value 202
- SecQOPIntegrityAndConfidentiality enumeration value 202
- SecQOPIntegrity enumeration value 202
- SecQOPNoProtection enumeration value 202
- SecTargetSecureInvocation policy 112, 200
- secure associations
 - client behavior 110
 - definition 104
 - TLS_Coloc interceptor 104
- secure_client_with_no_cert configuration
 - sample 162
- secure hash algorithms 121, 122
- secure invocation policy 107, 200
- secure_server_no_client_auth configuration 39
- secure_server_no_client_auth configuration
 - sample 162
- Secure Sockets Layer, *See* SSL
- Security 323
- security algorithms
 - and cipher suites 121
- security attribute service context 150, 154
- SecurityAttributeType type 221
- security capsule
 - and principal authenticator 211
 - credentials sharing 141, 168, 212
- security handshake
 - cipher suites 119
 - SSL/TLS 133, 137
- security_label authorization method ID 141
- SecurityManager interface 212, 215, 220
 - and credentials 222
 - retrieving own credentials 224
- Security recommendations 323
- self-signed certificate 85
- semi-secure applications
 - and NoProtection 117
- SEMI_SECURE servers 108
- serial number 81, 237
- server_binding_list configuration variable 50, 184
 - and CSv2 authentication 164
- server domain name
 - and CSv2 authentication over transport 165
- server_domain_name configuration variable
 - iSF, ignored by 52
- server-side policies 196
- server_version_policy
 - IIOp 297
- service contexts
 - security attribute 150, 154
 - _set_policy_overrides() operation 196
 - set_policy_overrides() operation 198, 241
 - and invocation credentials 204
 - SHA 122
 - SHA1 108
 - SHA256 108
 - shared credentials 141, 168, 212
 - signing certificates 80
 - Specifying ciphersuites 119
 - SSL/TLS
 - association options
 - setting 106
 - cipher suites 119
 - client configuration 40
 - colocated invocations 104
 - encryption algorithm 121
 - fixed ports 46
 - IIOp_TLS interceptor 41
 - key exchange algorithm 121
 - logging 41
 - mechanism policy 126
 - mixed configurations 44
 - orb_plugins list 41
 - principal sponsor
 - disabling 41
 - enabling 43
 - protocol_version configuration variable 126
 - secure associations 104
 - secure client, definition 36
 - secure hash algorithm 121
 - secure hash algorithms 122
 - secure invocation policy 107
 - secure server, definition 38
 - securing communications 36
 - security handshake 133, 137
 - selecting a toolkit, C++ 255
 - semi-secure client
 - IIOp plug-in 41
 - semi-secure client, definition 37
 - semi-secure server, definition 38
 - server configuration 42
 - terminology 36
 - TLS session 104
 - SSL/TLS policies 199
 - SSL/TLS principal sponsor
 - and CSv2 authentication over transport 163
 - SSL/TLS toolkits 100
 - stash file 95

- subject DN
 - and identity tokens 181
 - subject name 237
 - supported security features 201
 - Symmetric cryptography 28
 - symmetric encryption algorithms 121
- T**
- Target
 - choosing behavior 112
 - target and client authentication 136
 - example configuration 138
 - target authentication 133
 - target authentication only
 - example 135
 - target credentials
 - availability of 227
 - definition 221
 - interaction with rebind policy 227
 - retrieving 226
 - C++ example 228
 - SSL/TLS
 - C++ example 229
 - parsing 229
 - TargetCredentials interface 221, 227
 - Orbix-specific 222
 - target secure invocation policy
 - HTTPS 112
 - IIOPTLS 112
 - TargetSecureInvocationPolicy policy 107
 - TCP policies
 - delay connections 298
 - receive buffer size 298
 - terminology
 - SSL/TLS
 - secure client, definition 36
 - secure server, definition 38
 - semi-secure client, definition 37
 - semi-secure server, definition 38
 - SSL/TLS samples 36
 - three-tier scenario description 54
 - TLS
 - authentication 24
 - handshake 25
 - how provides security 24
 - integrity 29
 - TLS_CERT_CONSTRAINTS_POLICY policy type 242
 - TLS_Coloc interceptor 104
 - TLSCredentials interface 212, 225, 247
 - TLSTargetCredentials interface 232
 - TLS session
 - definition 104
 - TLSTargetCredentials interface
 - parsing target credentials 229
 - tokens
 - client authentication 160
 - toolkit replaceability 100
 - enabling JCE architecture 256
 - selecting the toolkit, C++ 255
 - Transport Layer Security, *See* TLS
 - triple DES 121
 - trusted CAs 86
 - trust in client
 - by programming, SSL/TLS 203
 - trust in target
 - by programming, SSL/TLS 203
- U**
- use_jsse_tk configuration variable 256
 - user key ring
 - adding certificates 92
 - creating 92
 - listing contents 92
 - username and password
 - CSlv2 168
- V**
- validate_cert_chain() operation 244
- W**
- well-known addressing policy 47
 - WellKnownAddressingPolicy policy 46
- X**
- X.509
 - and PKCS#12 file 139
 - certificates. *See* certificates
 - Extension interface 238
 - ExtensionList interface 238
 - extensions 238
 - public key encryption 121
 - v3 extensions 237, 238
 - X.509 certificate
 - contents 237
 - definition 80
 - X.509 certificates 79

INDEX

X509CertChain interface 247
X509CertificateFactory interface 247
X509Cert interface 247

Z

z/OS
 interoperability with 126