IONA®

Orbix®

Mainframe Security Guide

Version 6.2, May 2005

Making Software Work Together™

Updated: 13-May-2005

Contents

List of Tables

List of Figures

Preface

What is Covered in this Book

This book is a guide to administering and programming secure applications in Orbix, covering both secure CORBA applications.

The IONA 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 "CSIv2 Administration"

This part explains how to configure and manage CSIv2 in detail.

Part V **"CORBA Security Programming"**

This part explains how to program the SSL/TLS and CSIv2 APIs in your security-aware CORBA applications.

Appendices

The appendices list further technical details.

Related Documentation

The *Orbix 2000 Programmer's Guide C++ Edition* and *Orbix 2000 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 http://www.iona.com/docs.

Additional Resources for Information

The IONA knowledge base (http://www.iona.com/support/knowledge_base/index.xml) contains helpful articles, written by IONA experts, about Orbix and other products. You can access the knowledge base at the following location:

The IONA update center (http://www.iona.com/support/updates/index.xml) contains the latest releases and patches for IONA products.

If you need help with this or any other IONA products, contact IONA at `support@iona.com`. Comments on IONA documentation can be sent to docs-support@iona.com .

Security Alert Mailing List

There is a mailing list for customers to receive security alerts associated with IONA's products. The mail alias is `security-alert@iona.com`.

To subscribe, send an e-mail to `listserver@iona.com`. Leave the email `Subject` field blank and, in the body of the email, type:

`subscribe security-alert` *YourEmailAddress*

To unsubscribe, type:

`unsubscribe security-alert` *YourEmailAddress*

Note: Please do not try to post queries to this email alias; it has been set up only to notify you of security alerts.

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 DOS, Windows NT, Windows 95, or Windows 98 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 I

Introducing Security

In this part

This part contains the following chapters:

CHAPTER 1

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

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

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, which can service remote requests for security-related functions.

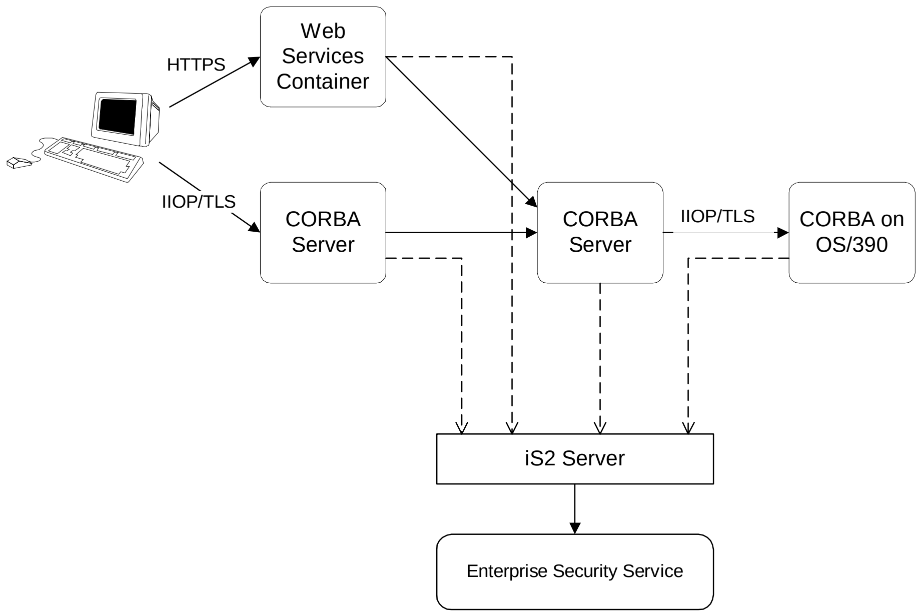


Figure 1: *Example System with a Standalone Orbix Security Service*

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 application, 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 11.

Propagating security credentials

The example in Figure 1 on page 6 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

iSF client API

The GSP plug-in access the Orbix security service through the iSF client API, which is a private IONA-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

IONA 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

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?

IONA'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.

CSIv2

The Common Secure Interoperability (CSIv2) plug-in provides support for authentication based on a username and password. The CSIv2 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.

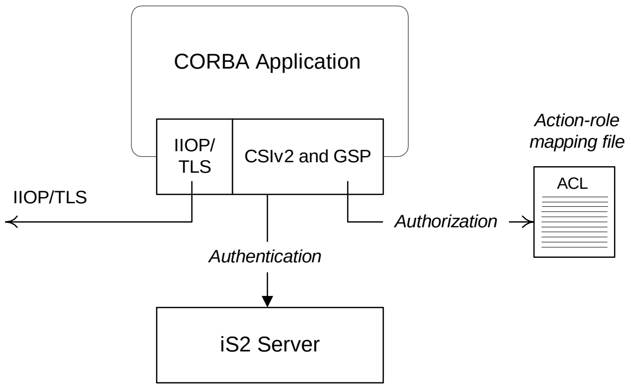


Figure 2: *Security Plug-Ins in a CORBA Application*

IIOP/TLS plug-in in CORBA a application

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

CSIv2 plug-in in a CORBA application

The CSIv2 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

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 485.

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.

See “iS2 Configuration” on page 513 for details.

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.

CHAPTER 2

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?

Security plug-ins

Orbix provides the core security infrastructure to a distributed system based on IONA'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 DES, Triple DES, RC4, IDEA).
- Message integrity (based on SHA1, MD5).
- 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

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 *certificate* 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 the Data Encryption Standard (DES) and RC4.

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.

Part II

Orbix Security Framework Administration

In this part

This part contains the following chapters:

Securing Applications and Services

This chapter describes how to enable security in the context of the IONA 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

Overview

Many of the examples in this chapter use the IONA security framework (iSF), which requires access to the iS2 server. Because Orbix Mainframe 6.0 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 IONA'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

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.

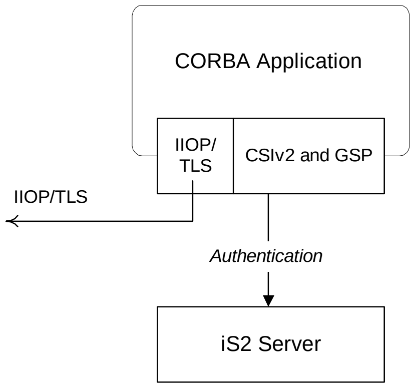


Figure 3: *A Secure CORBA Application within the iSF*

Security plug-ins

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

- IIOP/TLS plug-in
- CSIv2 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 38 for details on how to enable IIOP/TLS in a CORBA application.

CSIv2 plug-in (Java only)

The CSIv2 plug-in, `csi`, provides a mechanism for propagating username/password credentials between CORBA applications. When the CSIv2 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 CSIv2 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 291 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
`secure_client`	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`	The type of connection opened by the client depends on the disposition of the server: • 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 `binding:client_binding_list`. ♦ If, in the client's `binding:client_binding_list`, a binding with the `IIOP` interceptor appears before a binding with the `IIOP_TLS` interceptor, an insecure connection is established. ♦ Conversely, if a binding with the `IIOP_TLS` interceptor appears before a binding with the `IIOP` 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 = "SSL_V3";
      policies:mechanism_policy:ciphersuites =
     ["RSA_WITH_RC4_128_SHA", "RSA_WITH_RC4_128_MD5"];

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"];
      };
  };
  ...
```

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 IONA 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 50.

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

   If you plan to use the full IONA security framework, you should use the `binding:client_binding_list` as shown in “Client configuration” on page 51 instead.

3. The SSL/TLS mechanism policy specifies the default security protocol version and the available cipher suites—see “Specifying Cipher Suites” on page 103.

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         principal_sponsor:use_principal_sponsor = "true";
4         principal_sponsor:auth_method_id = "security_label";
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";
```

Example 2: *Sample SSL/TLS Server Configuration*

```
        # 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 42
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 119.
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 119.

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 85 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 44. 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 IIOP) 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 87.
- "Configuring SSL/TLS Authentication" on page 111.

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 IIOP/TLS 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 34.

Two-tier CORBA system

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

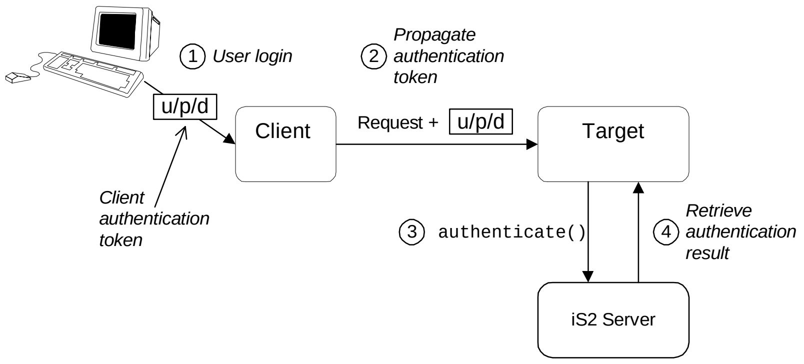

1 User login
2 Propagate authentication token
u/p/d
Client
Request + u/p/d
Target
Client authentication token
3 authenticate()
4 Retrieve authentication result
iS2 Server

Figure 4: *Two-Tier CORBA System in the iSF*

Scenario description

The scenario shown in Figure 4 on page 50 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 50 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.
        ...
        # Common iSF configuration settings.
2       orb_plugins = ["local_log_stream", "iiop_profile", "giop",
       "iiop_tls", "ots", "gsp"];
3       binding:client_binding_list = ["GIOP+EGMIOP",
       "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"];
4       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 / 6 / 7 (callout numbers beside code)

```
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 38 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. Your 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. Your 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 38.
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 148.

Target configuration

The CORBA target server from Figure 4 on page 50 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.
          ...
          # Specific iSF configuration settings.
2             policies:csi:auth_over_transport:target_supports =
      ["EstablishTrustInClient"];
3             policies:csi:auth_over_transport:target_requires =
      ["EstablishTrustInClient"];
4             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 38.
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 34.

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.

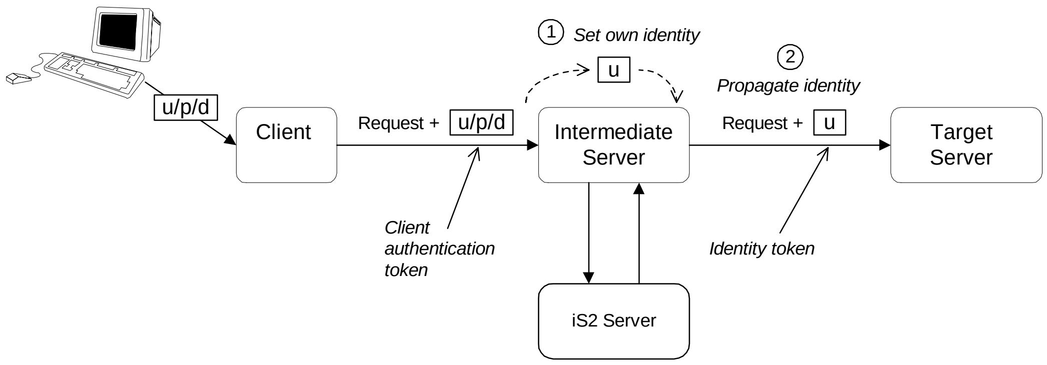


Figure 5: *Three-Tier CORBA System in the iSF*

Scenario description

The second stage of the scenario shown in Figure 5 on page 55 (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.

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 51.

Intermediate configuration

The CORBA intermediate server from Figure 5 on page 55 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 = [ ... ];
    ...
    my_three_tier_intermediate {
1       # Specific SSL/TLS configuration settings.
        ...
        # Specific iSF configuration settings.
```

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

```
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 38.
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 55 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       ...
2     policies:iiop_tls:target_secure_invocation_policy:requires
   = ["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 38.
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 124.

> **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 35.

Caching of Credentials

Overview

To improve the performance of servers within the IONA 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 IONA 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.

Part III

SSL/TLS Administration

In this part

This part contains the following chapters:

CHAPTER 4

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

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 IONA’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 67.

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 IONA Professional Services for more details:

http://www.iona.com/info/services/consulting/welcome.htm

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 7 to your Orbix configuration (the toolkit must be specified explicitly, because Orbix would select the Baltimore toolkit by default).

Example 7: *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).

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?

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 Appendix C on page 529 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

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 6 shows an example of a simple certificate chain.

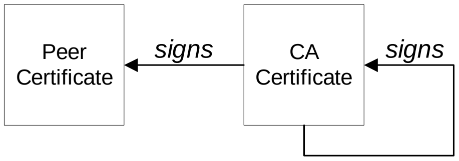


Figure 6: *A Certificate Chain of Depth 2*

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 IONA Technologies, which in turn is signed by a self-signed commercial CA. Figure 7 shows what this certificate chain looks like.

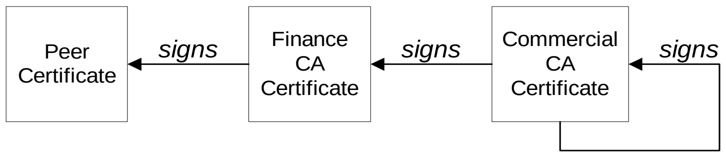


Figure 7: *A Certificate Chain of Depth 3*

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:iiop_tls:max_chain_length_policy` and `policies:https:max_chain_length_policy` configuration variables for IIOP/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 Guide and Reference*. 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

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 Orbix2000 on other platform, 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 IONA 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
...
...
oudXbfbjlQZQ+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 Guide and Reference*
- *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 79.

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 *System Secure Sockets Layer - Programming Guide and Reference* 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 *System Secure Sockets Layer - Programming Guide and Reference* for details of how to create a password stash file.

CHAPTER 6

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

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 94 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 96 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 103 for details.

Figure 8 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.

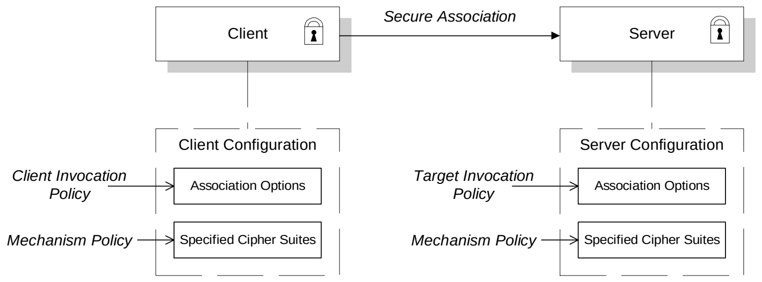


Figure 8: *Configuration of a Secure Association*

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

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 MD5, 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 *ASPInstallDir*`/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.

IIOP/TLS configuration

You can set this policy for IIOP/TLS connections through the following configuration variables:

`policies:iiop_tls:client_secure_invocation_policy:requires`
: Specifies the minimum security features that the client requires to establish an IIOP/TLS connection.

`policies:iiop_tls:client_secure_invocation_policy:supports`
: Specifies the security features that the client is able to support on IIOP/TLS 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 92 and Appendix C on page 287.

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.

IIOP/TLS configuration

You can set the target secure invocation policy for IIOP/TLS connections through the following configuration variables:

`policies:iiop_tls:target_secure_invocation_policy:requires`
: Specifies the minimum security features that your targets require, before they accept an IIOP/TLS connection.

`policies:iiop_tls:target_secure_invocation_policy:supports`
: Specifies the security features that your targets are able to support on IIOP/TLS 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 92 and Appendix C on page 287.

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 `demos.tls` scopes in the *DomainName*`.cfg` configuration file. In `demos.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 62.

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 108).
- 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 assocation 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...
`EstablishTrustInTarget`	`policies:client_secure_invocation_policy:supports` `policies:client_secure_invocation_policy:requires`	`policies:target_secure_invocation_policy:supports`
`EstablishTrustInClient`	`policies:client_secure_invocation_policy:supports`	`policies:target_secure_invocation_policy:supports` `policies:target_secure_invocation_policy:requires`

Note: The SSL/TLS client authentication step can also be affected by the `policies:allow_unauthenticated_clients_policy` configuration variable. See "policies Namespace" on page 498.

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 109). 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`	`policies:client_secure_invocation_policy:supports` `policies:client_secure_invocation_policy:requires`	`policies:target_secure_invocation_policy:supports` `policies:target_secure_invocation_policy:requires`

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`	`policies:client_secure_invocation_policy:supports` `policies:client_secure_invocation_policy:requires`	`policies:target_secure_invocation_policy:supports` `policies:target_secure_invocation_policy:requires`

References

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

- "Securing Communications with SSL/TLS" on page 62.
- The `demos.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

Orbix cipher suites

The following cipher suites are supported by Orbix:

- Null encryption, integrity-only ciphers:
  ```
  RSA_WITH_NULL_MD5
  RSA_WITH_NULL_SHA
  ```
- Standard ciphers
  ```
  RSA_EXPORT_WITH_RC4_40_MD5
  RSA_WITH_RC4_128_MD5
  RSA_WITH_RC4_128_SHA
  RSA_EXPORT_WITH_RC2_CBC_40_MD5
  RSA_WITH_DES_CBC_SHA
  RSA_WITH_3DES_EDE_CBC_SHA
  ```

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:

`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.

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.

Cipher suite definitions

The Orbix cipher suites are defined as follows:

Table 7: *Cipher Suite Definitions*

Cipher Suite	Key Exchange Algorithm	Encryption Algorithm	Secure Hash Algorithm	Exportable?
`RSA_WITH_NULL_MD5`	`RSA`	`NULL`	`MD5`	*yes*
`RSA_WITH_NULL_SHA`	`RSA`	`NULL`	`SHA`	*yes*
`RSA_EXPORT_WITH_RC4_40_MD5`	`RSA_EXPORT`	`RC4_40`	`MD5`	*yes*
`RSA_WITH_RC4_128_MD5`	`RSA`	`RC4_128`	`MD5`	*no*
`RSA_WITH_RC4_128_SHA`	`RSA`	`RC4_128`	`SHA`	*no*
`RSA_EXPORT_WITH_RC2_CBC_40_MD5`	`RSA_EXPORT`	`RC2_CBC_40`	`MD5`	*yes*
`RSA_WITH_DES_CBC_SHA`	`RSA`	`DES_CBC`	`SHA`	*no*
`RSA_WITH_3DES_EDE_CBC_SHA`	`RSA`	`3DES_EDE_CBC`	`SHA`	*no*

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

- Whether SSL or TLS is used, and
- Which specific cipher suites are to be used.

The protocol_version configuration variable

You can specify whether SSL or TLS is used with a transport protocol by setting 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 = "SSL_V3";
```

You can set the `protocol_version` configuration variable to one of the following alternatives:

```
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
```

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

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 =
 ["RSA_WITH_NULL_MD5",
  "RSA_WITH_NULL_SHA",
  "RSA_EXPORT_WITH_RC4_40_MD5",
  "RSA_WITH_RC4_128_MD5" ];
```

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:

- Null encryption, integrity only ciphers:

  ```
  RSA_WITH_NULL_MD5,
  RSA_WITH_NULL_SHA
  ```
- Standard ciphers

  ```
  RSA_EXPORT_WITH_RC4_40_MD5,
  RSA_WITH_RC4_128_MD5,
  RSA_WITH_RC4_128_SHA,
  RSA_EXPORT_WITH_RC2_CBC_40_MD5
  RSA_WITH_DES_CBC_SHA,
  RSA_WITH_3DES_EDE_CBC_SHA
  ```

Default values

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

```
RSA_WITH_RC4_128_SHA,
RSA_WITH_RC4_128_MD5,
RSA_WITH_3DES_EDE_CBC_SHA,
RSA_WITH_DES_CBC_SHA
```

Constraints Imposed on Cipher Suites

Effective cipher suites

Figure 9 shows that cipher suites initially specified in the configuration are *not* necessarily made available to the application. Orbix checks each cipher suite for compatibility with the specified association options and, if necessary, reduces the size of the list to produce a list of *effective cipher suites*.

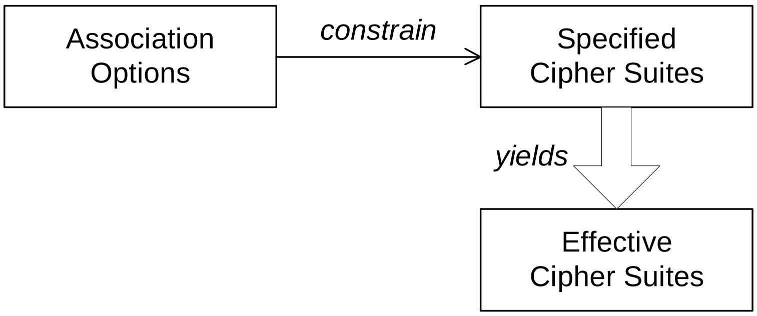


Figure 9: *Constraining the List of Cipher Suites*

Required and supported association options

For example, in the context of the IIOP/TLS protocol the list of cipher suites is affected by the following configuration options:

- *Required association options*—as listed in `policies:iiop_tls:client_secure_invocation_policy:requires` on the client side, or `policies:iiop_tls:target_secure_invocation_policy:requires` on the server side.
- *Supported association options*—as listed in `policies:iiop_tls:client_secure_invocation_policy:supports` on the client side, or `policies:iiop_tls:target_secure_invocation_policy:supports` on the server side.

Cipher suite compatibility table

Use Table 8 to determine whether or not a particular cipher suite is compatible with your association options.

Table 8: *Association Options Supported by Cipher Suites*

Cipher Suite	Supported Association Options
RSA_WITH_NULL_MD5	Integrity, DetectReplay, DetectMisordering
RSA_WITH_NULL_SHA	Integrity, DetectReplay, DetectMisordering
RSA_EXPORT_WITH_RC4_40_MD5	Integrity, DetectReplay, DetectMisordering, Confidentiality
RSA_WITH_RC4_128_MD5	Integrity, DetectReplay, DetectMisordering, Confidentiality
RSA_WITH_RC4_128_SHA	Integrity, DetectReplay, DetectMisordering, Confidentiality
RSA_EXPORT_WITH_RC2_CBC_40_MD5	Integrity, DetectReplay, DetectMisordering, Confidentiality
RSA_WITH_DES_CBC_SHA	Integrity, DetectReplay, DetectMisordering, Confidentiality
RSA_WITH_3DES_EDE_CBC_SHA	Integrity, DetectReplay, DetectMisordering, Confidentiality

Determining compatibility

The following algorithm is applied to the initial list of cipher suites:

1. For the purposes of the algorithm, ignore the `EstablishTrustInClient` and `EstablishTrustInTarget` association options. These options have no effect on the list of cipher suites.
2. From the initial list, remove any cipher suite whose supported association options (see Table 8) do not satisfy the configured required association options.
3. From the remaining list, remove any cipher suite that supports an option (see Table 8) not included in the configured supported association options.

No suitable cipher suites available

If no suitable cipher suites are available as a result of incorrect configuration, no communications will be possible and an exception will be raised. Logging also provides more details on what went wrong.

Example

For example, specifying a cipher suite such as `RSA_WITH_RC4_128_MD5` that supports `Confidentiality`, `Integrity`, `DetectReplay`, `DetectMisordering`, `EstablishTrustInTarget` (and optionally `EstablishTrustInClient`) but specifying a `secure_invocation_policy` that supports only a subset of those features results in that cipher suite being ignored.

CHAPTER 7

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

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 87. 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

Overview

When an application is configured for target authentication only, the target authenticates itself to the client but the client is not authentic to the target object—see Figure 10.

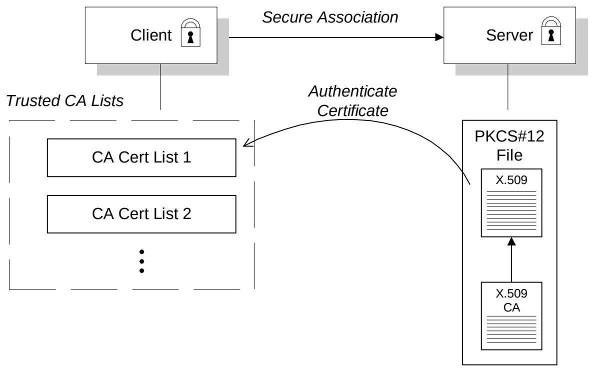


Figure 10: *Target Authentication Only*

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 119.
- 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 10. 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 E2A support target authentication.

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.

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 E2A support target authentication.

Example of target authentication only

The following sample extract from an Orbix E2A 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 = "SSL_V3";
policies:iiop_tls:mechanism_policy:ciphersuites =
    ["RSA_WITH_RC4_128_SHA", "RSA_WITH_RC4_128_MD5"];

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 11. In this case, the server and the client each require an X.509 certificate for the security handshake.

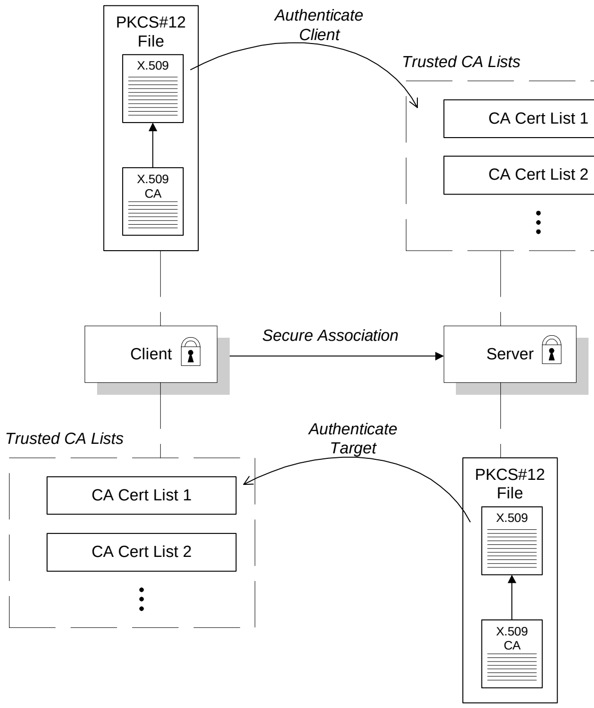


Figure 11: *Target and Client Authentication*

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 119.
- 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 1.

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.

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.

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 E2A 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 = "SSL_V3";
policies:iiop_tls:mechanism_policy:ciphersuites =
   ["RSA_WITH_RC4_128_SHA", "RSA_WITH_RC4_128_MD5"];

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 189.

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 72).
- 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 12 shows the typical elements in a PKCS#12 file.

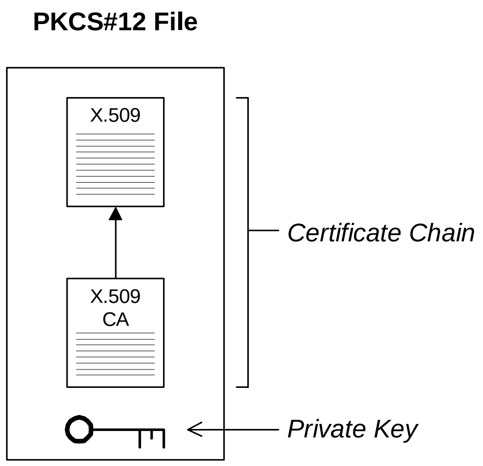


Figure 12: *Elements in a PKCS#12 File*

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 189). 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 85.

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 79) 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 69, 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

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=IONA,C=Ireland,ST=Dublin,L=Ea
   rth","CN=Paul*,OU=SSLTEAM,O=IONA,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 529.

Part IV
CSIv2 Administration

In this part

This part contains the following chapters:

CHAPTER 8

Introduction to CSIv2

CSIv2 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 CSIv2 to transmit usernames and passwords, and asserted identities between applications.

In this chapter

This chapter discusses the following topics:

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.

CSIv2 identity assertion mechanism

The CSIv2 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 CSIv2

CSIv2 is applicable to both CORBA technology. CSIv2 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

Overview

Figure 13 shows a basic CSIv2 scenario where a CORBA client and a CORBA server are configured to use the CSIv2 authentication over transport mechanism.

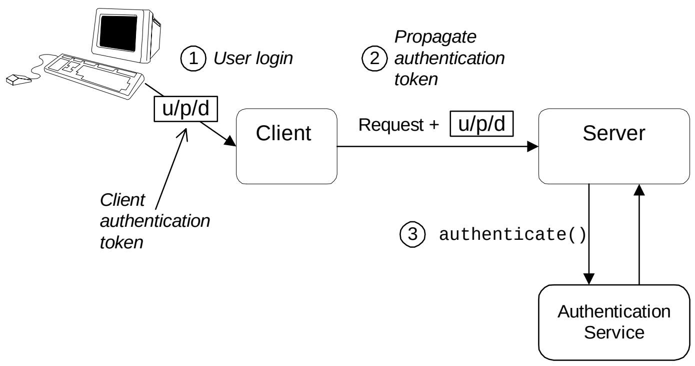


Figure 13: *Basic CSIv2 Authentication over Transport Scenario*

Scenario description

The scenario shown in Figure 13 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 137.

CSIv2 Identity Assertion Scenario

Overview

Figure 14 shows a basic CSIv2 scenario where a client and an intermediate server are configured to use the CSIv2 authentication over transport mechanism, and the intermediate server and a target server are configured to use the CSIv2 identity assertion mechanism. In this scenario, the client invokes on the intermediate server, which then invokes on the target server.

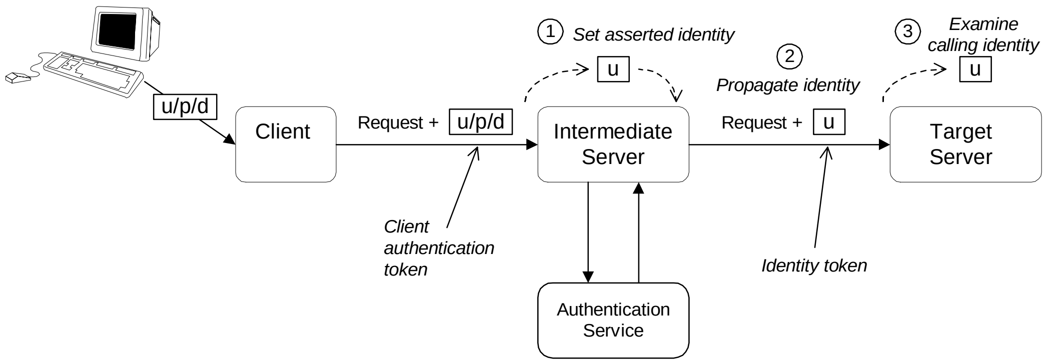


Figure 14: *Basic CSIv2 Identity Assertion Scenario*

Scenario description

The second stage of the scenario shown in Figure 14 (intermediate server invokes an operation on the target server) can be described as follows:

Stage	Description
1	The intermediate server can set the identity that will be asserted to the target in one of two ways: • Implicitly—if the execution context has an associated CSIv2 received credentials, the intermediate server extracts the user identity from the received credentials, or • Explicitly—by programming.

Stage	Description
2	When the intermediate server makes a remote invocation on the target server, CSIv2 transmits the user identity data to the server in a security attribute service context.
3	The target server can access the propagated user identity programmatically (by extracting it from a `SecurityLevel2::ReceivedCredentials` object).

More details

For more details about identity assertion, see "Configuring CSIv2 Identity Assertion" on page 157.

CHAPTER 9

Configuring CSIv2 Authentication over Transport

This chapter explains the concepts underlying the CSIv2 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:

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 IONA'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.

CSIv2 scenario

Figure 15 shows a typical scenario for CSIv2 authentication over transport:

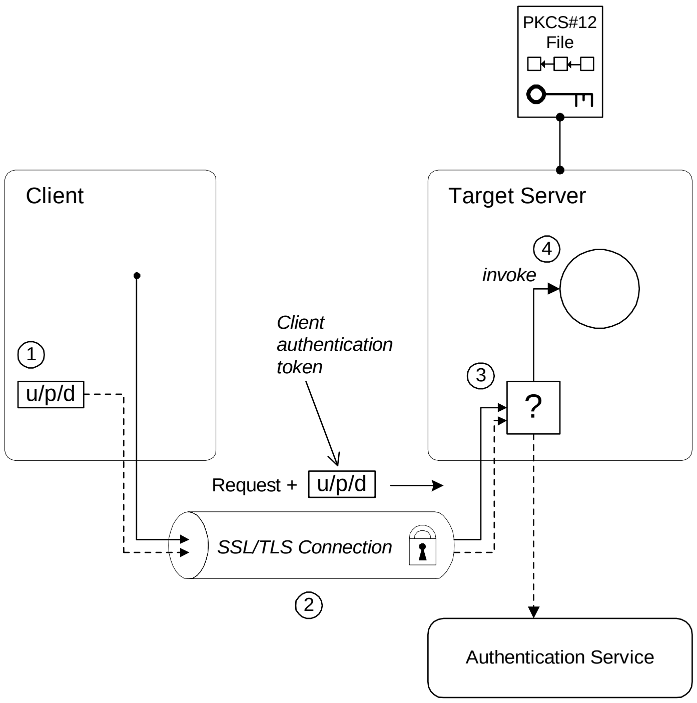


Figure 15: *CSIv2 Authentication Over Transport Scenario*

How CSIv2 authentication over transport proceeds

As shown in Figure 15 on page 139, 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 142 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 *CSIv2 principal sponsor* (which prompts the user to enter the username/password and domain). See “Providing a Username and Password” on page 148.

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 147.

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 15 on page 139, 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 CSIv2 principal sponsor (see "CSIv2 principal sponsor" on page 148). It is possible, therefore, to enable both of the principal sponsors within the same application.

References

See "Sample Configuration" on page 152 for a detailed example of the client and server SSL/TLS configuration.

See "SSL/TLS Administration" on page 257 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 142) and the CSIv2 principal sponsor (see "Providing a Username and Password" on page 148).

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.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"];
    ...
};
```

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 CSIv2 authentication service in Java.

When using CSIv2 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 CSIv2 authentication service programmatically.

See "Providing an Authentication Service" on page 147 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:

ASPInstallDir`/asp/`*Version*`/demos/corba/tls/csiv2/java/src/csiv2`

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).

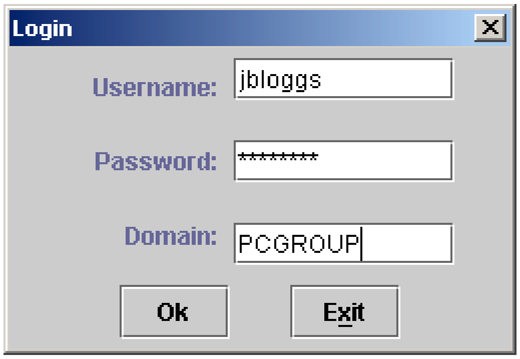
C++ Applications

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:



Username: jbloggs
Password: ********
Domain: PCGROUP

Figure 16: *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 197.

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 to 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 "CSIv2 Authentication Scenario" on page 138.

In this section

This section contains the following subsections:

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\6.0\etc\tls\x509\certs\demos\b
   ank_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";
    };
};
```

CHAPTER 10

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

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 17). 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.

CSIv2 scenario

Figure 17 shows a typical scenario for CSIv2 identity assertion:

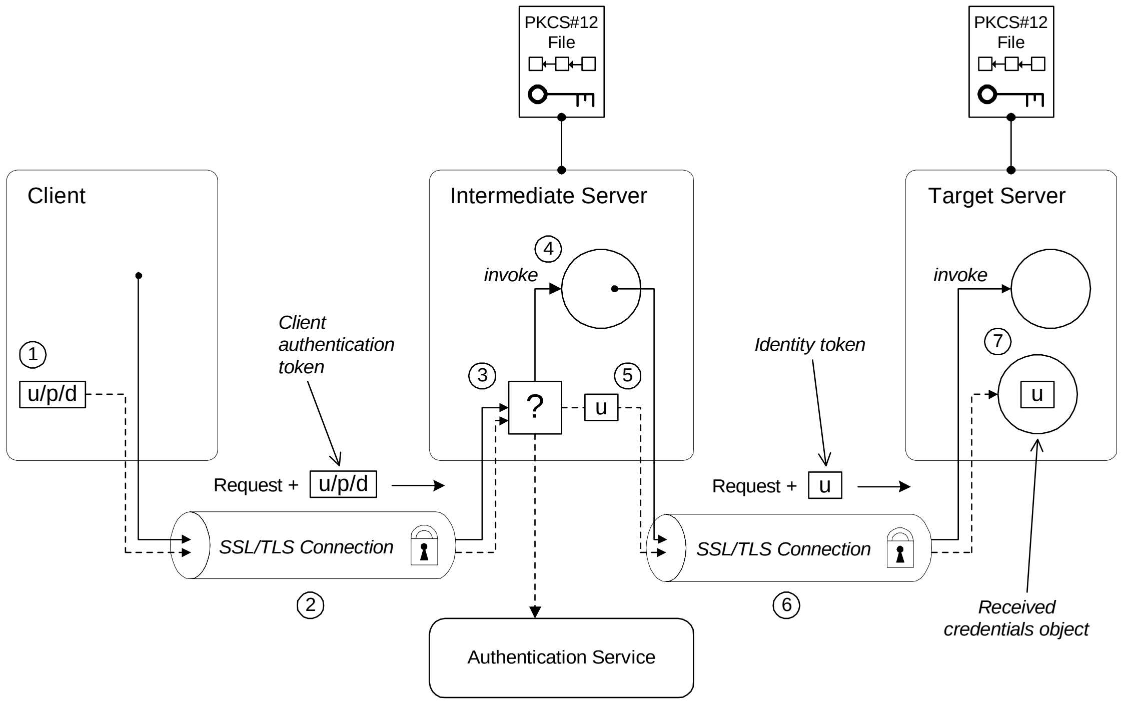


Figure 17: *CSIv2 Identity Assertion Scenario*

How CSIv2 identity assertion proceeds

As shown in Figure 17 on page 159, 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 *identity token* 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 209.

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 162 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 CSIv2 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 210.

SSL/TLS Prerequisites

Overview

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.

SSL/TLS mutual authentication

For the scenario depicted in Figure 17 on page 159, 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 169 for a detailed example of the SSL/TLS configuration in this scenario.

See "SSL/TLS Administration" on page 257 for complete details of configuring and administering SSL/TLS.

Setting certificate constraints

In the scenario depicted in Figure 17 on page 159, 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 124 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 119 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 148). 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 17 on page 159, 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 166.

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 "CSIv2 Identity Assertion Scenario" on page 158.

In this section

This section contains the following subsections:

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\6.0\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:iiop_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\6.0\etc\tls\x509\certs\demos\b
   ank_server.p12", "password=bankserverpass"];
        policies:csi:attribute_service:target_supports =
   ["IdentityAssertion"];
    };
};
```

Part V

CORBA Security Programming

In this part

This part contains the following chapters:

CHAPTER 11

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

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 9.

Table 9: *Policy Management Objects*

Policy Level	Policy Management Object
ORB	`CORBA::PolicyManager`
Thread	`CORBA::PolicyCurrent`
POA	`PortableServer::POA::create_POA()`
Client-side proxy	(*ObjectRef*)`._set_policy_overrides()`

C++ Example

The following C++ example shows how to set an SSL/TLS certificate constraints policy at the ORB level:

Example 8: *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"
       );

       any <<= cert_constraints;
4,5    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 177, 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

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 Namespace" on page 498).

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/6.0/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 90, 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 108.

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 `DetectReplay` 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 108.

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 CSIv2 Policies

Overview

This section gives a brief overview of the programmable CSIv2 policies. These programmable policies provide functionality equivalent to the CSIv2 configuration variables.

For complete details of the CSIv2 policies, see the description of the `IT_CSI` module in the *CORBA Programmer's Reference*.

CSIv2 policies

The following CSIv2 policies can be set programmatically:

- Client-side CSIv2 authentication policy.
- Server-side CSIv2 authentication policy.
- Client-side CSIv2 identity assertion policy.
- Server-side CSIv2 identity assertion policy.

Client-side CSIv2 authentication policy

You can set the client-side CSIv2 authentication policy to enable an application to send GSSUP username/password credentials over the wire in a GIOP service context. The programmable client-side CSIv2 authentication policy provides functionality equivalent to setting the following configuration variable:

```
policies:csi:auth_over_transport:client_supports
```

To create a client-side CSIv2 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 CSIv2 authentication policy

You can set the server-side CSIv2 authentication policy to enable an application to receive and authenticate GSSUP username/password credentials. The programmable server-side CSIv2 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 CSIv2 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 CSIv2 identity assertion policy

You can set the client-side CSIv2 identity assertion policy to enable an application to send a CSIv2 asserted identity over the wire in a GIOP service context. The programmable client-side CSIv2 identity assertion policy provides functionality equivalent to setting the following configuration variable:

```
policies:csi:attribute_service:client_supports
```

To create a client-side CSIv2 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 CSIv2 identity assertion policy

You can set the server-side CSIv2 identity assertion policy to enable an application to receive a CSIv2 asserted identity. The programmable server-side CSIv2 identity assertion policy provides functionality equivalent to setting the following configuration variable:

```
policies:csi:attribute_service:target_supports
```

To create a server-side CSIv2 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`.

CHAPTER 12

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

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

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 119.
- *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 `SecurityLevel2::SecurityManager` object.
2	Acquire a `SecurityLevel2::PrincipleAuthenticator` object from the security manager.
3	Call the `PrincipleAuthenticator::authenticate()` operation to authenticate the client principal and create a `SecurityLevel2::Credentials` own credentials object.
4	If more than one type of own credentials object is needed, call the `PrincipleAuthenticator::authenticate()` operation again with the appropriate arguments.

Types of credentials

Using the `PrincipalAuthenticator`, you can create the following types of credentials:

- SSL/TLS own credentials.
- CSIv2 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.

CSIv2 own credentials

The contents of a CSIv2 own credentials depends on the particular mechanism that is used, as follows:

- Username and password—if the CSIv2 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 9: *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 9: *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 `PrincipleAuthenticator` 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 10 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 10: *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;
1           obj = orb->resolve_initial_references(
                      "SecurityManager"
                  );
            security_manager_obj =
                SecurityLevel2::SecurityManager::_narrow(obj);
```

Example 10: *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;
            }

            username = CORBA::string_dup("<user_name>");
3           csi_gssup_auth_data.password =
                CORBA::string_dup("<password>");
            csi_gssup_auth_data.domain =
                CORBA::string_dup("<domain>");
4           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 10: *C++ Example of CSIv2 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

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 11.

Example 11: *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

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 E2A 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 12.

Example 12: *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 12 on page 207, 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

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 13.

Example 13: *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 13 on page 210) 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.
...
```

CHAPTER 13

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

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.

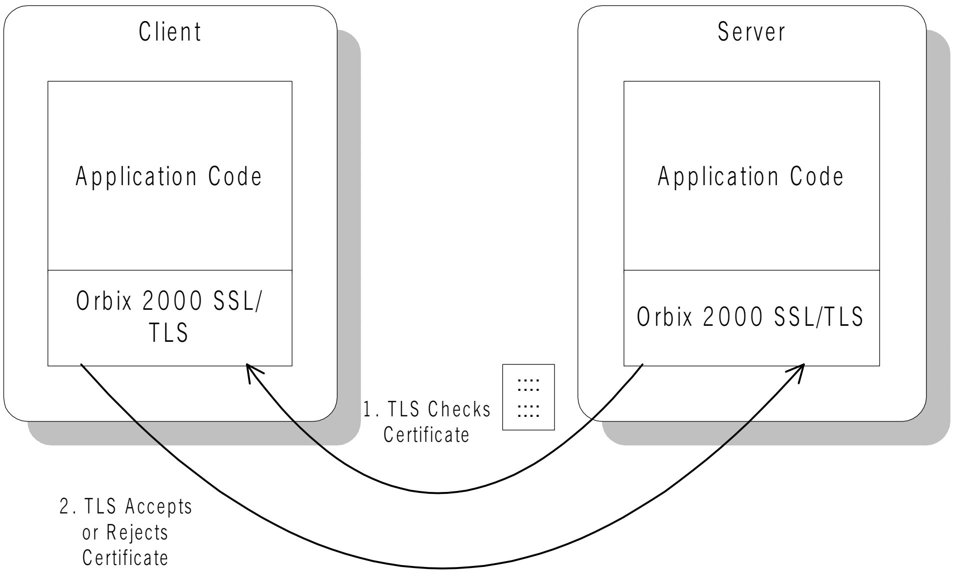


Figure 18: *Validating a Certificate*

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 19 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.

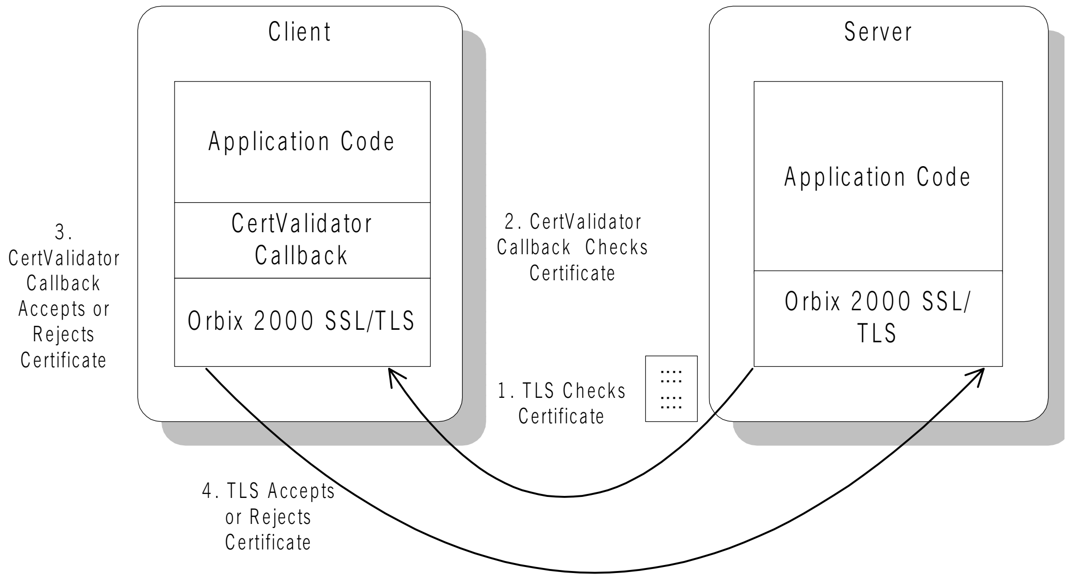

Client
Application Code
CertValidator Callback
Orbix 2000 SSL/TLS
Server
Application Code
Orbix 2000 SSL/TLS
1. TLS Checks Certificate
2. CertValidator Callback Checks Certificate
3. CertValidator Callback Accepts or Rejects Certificate
4. TLS Accepts or Rejects Certificate

Figure 19: *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 E2A 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*.

Orbix E2A ASP**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=IONA/OU=PD/CN=IONA
```

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:

`CertConstraintsPolicy`	Use this policy to apply conditions that peer X.509 certificates must meet to be accepted.
`CertificateValidatorPolicy`	Use this policy to create customized validations of peer certificate chains.

In this section

This section contains the following subsections:

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 124.

Setting the CertConstraintsPolicy by programming

When you specify a `CertConstraintsPolicy` object on an ORB programatically, 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 14: *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 an example custom validation function simply retrieves a name from a certificate:

Example 15: *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 "Orbix E2A ASPWorking with distinguished names in C++" on page 218) 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 16: *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 16: *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 Orbix2000 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 202.
- 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 For more information on this interface, see the *CORBA Programmer’s Reference*.

Security

This appendix describes variables used by the IONA Security Framework. The Orbix security infrastructure is highly configurable.

In this appendix

This appendix discusses the following topics:

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=IONA,C=Ireland,ST=Dublin,L=Ea
   rth","CN=Paul*,OU=SSLTEAM,O=IONA,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 the *Security Guide*.

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 Orbix. 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 default is the Baltimore 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 Orbix Java applications. If true, Orbix uses the JSSE/JCE architecture to implement SSL/TLS security; if false, Orbix 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):

- ClassName
- shlib_name

ClassName

ClassName specifies the Java class that implements the csi plugin. The default setting is:

```
plugins:csi:ClassName = "com.iona.corba.security.csi.CSIPlugin";
```

This configuration setting makes it possible for the Orbix core to load the plugin on demand. Internally, the Orbix core uses a Java class loader to load and instantiate the csi class. Plugin loading can be initiated either by including the csi in the orb_plugins list, or by associating the plugin with an initial reference.

shlib_name

shlib_name identifies the shared library (or DLL in Windows) containing the csi plugin implementation.

```
plugins:csi:shlib_name = "it_csi_prot";
```

The csi plug-in becomes associated with the it_csi_prot shared library, where it_csi_prot is the base name of the library. The library base name, it_csi_prot, is expanded in a platform-dependent manner to obtain the full name of the library file.

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_policy_enforcement_point`
- `authorization_policy_store_type`
- `authorization_realm`
- `ClassName`
- `enable_authorization`
- `enable_gssup_sso`
- `enable_user_id_logging`
- `enable_x509_sso`
- `enforce_secure_comms_to_sso_server`
- `enable_security_service_cert_authentication`
- `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 Orbix 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_policy_enforcement_point

Specifies whether access decisions should be made locally (based on cached ACL data) or delegated to the Orbix security service. This variable is meaningful only when the `authorization_policy_store_type` is set to `centralized`.

This configuration variable can have the following values:

- `local`—after retrieving and caching ACL data from the Orbix security service, the GSP plug-in consults only the local cache when making access decisions.
- `centralized`—this option is currently *not* implemented. If you set this option, the application will throw a `CORBA::NO_IMPLEMENT` system exception.

The default is `local`.

authorization_policy_store_type

Specifies whether ACL data should be stored locally (on the same host as the Orbix application) or centrally (on the same host as the Orbix security server). This configuration variable can have the following values:

- `local`—retrieves ACL data from the local file specified by the `plugins:gsp:action_role_mapping_file` configuration variable.
- `centralized`—retrieves ACL data from the Orbix security service. The Orbix security service must be configured to support centralized ACLs by editing the relevant properties in its `is2.properties` file.

The default is `local`.

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).

ClassName

`ClassName` specifies the Java class that implements the `gsp` plugin. This configuration setting makes it possible for the Orbix core to load the plugin on demand. Internally, the Orbix core uses a Java class loader to load and instantiate the `gsp` class. Plugin loading can be initiated either by including the `csi` in the `orb_plugins` list, or by associating the plugin with an initial reference.

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 Orbix security service.

Default is `false`.

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 231.

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 Orbix 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.iona.corba.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
- delay_credential_gathering_until_handshake
- enable_iiop_1_0_client_support
- incoming_connections:hard_limit
- incoming_connections:soft_limit
- outgoing_connections:hard_limit
- outgoing_connections:soft_limit
- 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 `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop: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 `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:buffer_pools:max_outgoing_buffers_in_pool` variable's value.

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 Orbix 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`, Orbix 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.

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.

tcp_listener:reincarnate_attempts

(Windows only)

`plugins:iiop_tls:tcp_listener:reincarnate_attempts` specifies the number of times that a Listener recreates its listener socket after recieving 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:iiop_tls: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:iiop_tls: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:locator

The plugins:locator namespace contains the following variable:

- iiop_tls:port

iiop_tls:port

Specifies the IP port number where the Orbix locator service listens for secure connections.

Note: This is only useful for applications that have a single TLS listener. For applications that have multiple TLS listeners, you need to programmatically specify the well-known addressing policy.

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

(Deprecated in favor of `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 an Orbix application deployed on the z/OS platform. When `true`, the Orbix application accepts V2 client hellos, but continues the handshake using either the SSL_V3 or TLS_V1 protocol. When `false`, the Orbix 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 shown in Table 10 can be specified in this list.

Table 10: *Mechanism Policy Cipher Suites*

Null Encryption, Integrity and Authentication Ciphers	Standard Ciphers
RSA_WITH_NULL_MD5	RSA_EXPORT_WITH_RC4_40_MD5
RSA_WITH_NULL_SHA	RSA_WITH_RC4_128_MD5
	RSA_WITH_RC4_128_SHA
	RSA_EXPORT_WITH_DES40_CBC_SHA
	RSA_WITH_DES_CBC_SHA
	RSA_WITH_3DES_EDE_CBC_SHA

If you do not specify the list of cipher suites explicitly, all of the null encryption ciphers are disabled and all of the non-export strength ciphers are supported by default.

mechanism_policy:protocol_version

(Deprecated in favor of `policies:iiop_tls: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 values `SSL_V3` and `TLS_V1`. 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 keyring of the HFS database).

policies:csi

The `policies:csi` namespace includes variables that specify settings for Common Secure Interoperability version 2 (CSIv2):

- `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 CSIv2 attribute service (principal propagation). The only assocation option that can be specified is `IdentityAssertion`. This policy is normally specified in an intermediate server so that it propagates CSIv2 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 assocation 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 assocation 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 (CSIv2 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 CSIv2 client about to open a connection to this server would check that the domain name in its own CSIv2 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 CSIv2 authorization over transport. The only assocation 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 CSIv2 authorization over transport. The only assocation 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`
- `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`
- `session_caching_policy`
- `target_secure_invocation_policy:requires`
- `target_secure_invocation_policy:supports`
- `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).

certificate_constraints_policy

A list of constraints applied to peer certificates—see "Applying Constraints to Certificates" on page 231 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 overides 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 Orbix server accepts V2 client hellos, but continues the handshake using either the SSL_V3 or TLS_V1 protocol. When `false`, the Orbix 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:

Table 11: *Mechanism Policy Cipher Suites*

Null Encryption, Integrity and Authentication Ciphers	Standard Ciphers
RSA_WITH_NULL_MD5	RSA_EXPORT_WITH_RC4_40_MD5
RSA_WITH_NULL_SHA	RSA_WITH_RC4_128_MD5
	RSA_WITH_RC4_128_SHA
	RSA_EXPORT_WITH_DES40_CBC_SHA
	RSA_WITH_DES_CBC_SHA
	RSA_WITH_3DES_EDE_CBC_SHA

If you do not specify the list of cipher suites explicitly, all of the null encryption ciphers are disabled and all of the non-export strength ciphers are supported by default.

mechanism_policy:protocol_version

This HTTPS-specific policy overides the generic `policies:mechanism_policy:protocol_version` policy.

Specifies the list of protocol versions used by a security capsule (ORB instance). Can include one or more of the following values:

`TLS_V1`
`SSL_V3`

The default setting is `SSL_V3` and `TLS_V1`.

For example:

```
policies:https:mechanism_policy:protocol_version = ["TLS_V1",
   "SSL_V3"];
```

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.

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/6.0/etc/tls/x509/ca/ca_list1.pem",
   "ASPInstallDir/asp/6.0/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.

policies:iiop_tls

The `policies:iiop_tls` namespace contains variables used to set IIOP-related policies for a secure environment. These setting affect the `iiop_tls` plugin. It contains the following variables:

- `allow_unauthenticated_clients_policy`
- `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`

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).

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 Orbix 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 *Orbix 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 *Orbix 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:iiop:server_version_policy="1.1";
```

The following `itadmin` command set this variable:

```
itadmin variable modify -type string -value "1.1"
   policies:iiop: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 *Orbix* 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 overides `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 overides the generic `policies:mechanism_policy:accept_v2_hellos` policy.

The `accept_v2_hellos` policy is a special setting that facilitates interoperability with an Orbix application deployed on the z/OS platform. Orbix 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 Orbix application accepts V2 client hellos, but continues the handshake using either the SSL_V3 or TLS_V1 protocol. When `false`, the Orbix 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 overides `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:

Table 12: *Mechanism Policy Cipher Suites*

Null Encryption, Integrity and Authentication Ciphers	Standard Ciphers
RSA_WITH_NULL_MD5	RSA_EXPORT_WITH_RC4_40_MD5
RSA_WITH_NULL_SHA	RSA_WITH_RC4_128_MD5
	RSA_WITH_RC4_128_SHA
	RSA_EXPORT_WITH_DES40_CBC_SHA

Table 12: *Mechanism Policy Cipher Suites*

Null Encryption, Integrity and Authentication Ciphers	Standard Ciphers
	RSA_WITH_DES_CBC_SHA
	RSA_WITH_3DES_EDE_CBC_SHA

If you do not specify the list of cipher suites explicitly, all of the null encryption ciphers are disabled and all of the non-export strength ciphers are supported by default.

mechanism_policy:protocol_version

This IIOP/TLS-specific policy overides the generic `policies:mechanism_policy:protocol_version` policy.

Specifies the list of protocol versions used by a security capsule (ORB instance). Can include one or more of the following values:

`TLS_V1`
`SSL_V3`
`SSL_V2V3` *(Deprecated)*

The default setting is `SSL_V3` and `TLS_V1`.

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 Orbix 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/configuration repository, 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` specifes 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 overides `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 *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

This policy overides `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 *Orbix 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 coresponds 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 coresponds to using the default size defined by the operating system.

trusted_ca_list_policy

This policy overides 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 =
   ["ASPInstallDir/asp/6.0/etc/tls/x509/ca/ca_list1.pem",
   "ASPInstallDir/asp/6.0/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.

principal_sponsor

The principal_sponsor namespace stores configuration information to be used when obtaining credentials. Orbix provides an implementation of a principal sponsor that creates credentials for applications automatically. The principal sponsor automatically calls the authenticate() operation on the PrincipalAuthenticator object after determining the data to supply.

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:

`pkcs12_file`	The authentication method uses a PKCS#12 file. Not supported in z/OS.
`security_label`	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`:

`filename`	A PKCS#12 file that contains a certificate chain and private key—*required*.
`password`	A password for the private key—*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.
`password_file`	The name of a file containing the password for the private key—*optional*. 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`:

`label`	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.iona.corba.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`:

`username`	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`.
`password`	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.
`domain`	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 `domain` 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

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=IONA Technologies,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 13 shows a selection of the attribute types that you are most likely to encounter:

Table 13: *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 13). 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

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 14 shows how the options affect client and target policies:

Table 14: *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 14: *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.

APPENDIX D

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

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 62.

Client configurations

The following client configurations are included in Example 17:

- demos.tls.secure_client_with_cert
- demos.tls.semi_secure_client_with_cert

Server configurations

The following server configurations are included in Example 17:

- 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 17 shows a variety of sample SSL/TLS configurations that you can copy or adapt for use in your own applications.

Example 17: *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 17: *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"];

            policies:client_secure_invocation_policy:supports =
```

Example 17: *SSL/TLS Configurations on the z/OS Platform*

```
["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";
        };

        semi_secure_server_no_client_auth
        {
```

Example 17: *SSL/TLS Configurations on the z/OS Platform*

```
            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

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).

Index