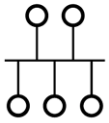

Orbix®

Security Guide

Version 6.2, December 2004

Making Software Work Together™

Updated: 25-Jul-2005

Contents

Part II Orbix Security Framework Administration

Part IV CSIv2 Administration

Part V CORBA Security Programming

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 *CORBA Programmer's Guide* and *CORBA Programmer's Reference* provide details about developing Orbix applications in C++ and Java.

The complete set of documentation for Orbix E2A ASP is available online at:

http://www.iona.com/docs/e2a/asp/6.0

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

Getting Started with Security

This chapter focuses on getting some security demonstrations up and running quickly. The details and background of the various security features are not discussed at this stage.

In this chapter

This chapter discusses the following topics:

Creating a Secure Domain

Overview

This section describes how to create a secure configuration domain, `secure`, which is required for the security demonstrations. This domain deploys a minimal set of Orbix services.

WARNING: The secure domain created using this procedure is *not* fully secure, because the X.509 certificates used in this domain are insecure demonstration certificates. This secure domain *must* be properly customized before deploying in a production environment.

Prerequisites

Before creating a secure domain, the following prerequisites must be satisfied:

- Your license allows you to use the security features of Orbix.
- Some basic system variables are set up (in particular, the `IT_PRODUCT_DIR`, `IT_LICENSE_FILE`, and `PATH` variables).

Fore more details, please consult the *Installation Guide*.

Licensing

The location of the license file, `licenses.txt`, is specified by the `IT_LICENSE_FILE` system variable. If this system variable is not already set in your environment, you can set it now.

Steps

To create a secure configuration domain, `secure`, perform the following steps:

1. Run itconfigure.
2. Choose the domain type.
3. Specify service startup options.
4. Specify security settings.
5. Specify fault tolerance settings.
6. Select services.
7. Confirm choices.
8. Finish configuration.

Run itconfigure

To begin creating a new configuration domain, enter `itconfigure` at a command prompt. An **Orbix Configuration Welcome** dialog box appears, as shown in Figure 1.

Select **Create a new domain** and click **OK**.

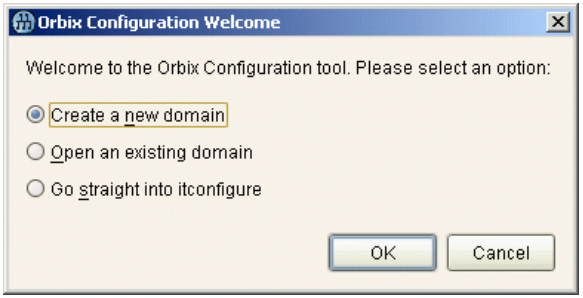

Figure 1: *The Orbix Configuration Welcome Dialog Box*

Choose the domain type

A **Domain Type** window appears, as shown in Figure 2.

In the **Configuration Domain Name** text field, type secure. Under **Configuration Domain Type**, click the **Select Services** radiobutton.

Click **Next>** to continue.

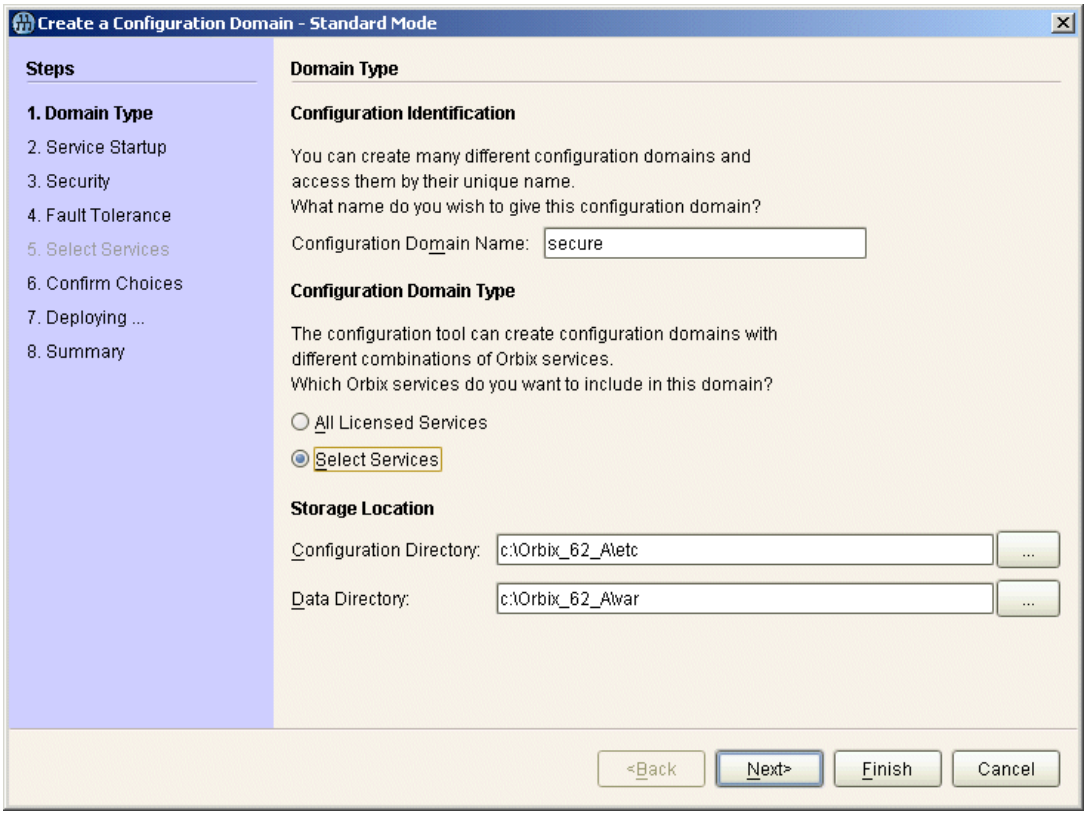

Figure 2: *The Domain Type Window*

Specify service startup options

A **Service Startup** window appears, as shown in Figure 3.

You can leave the settings in this Window at their defaults.

Click **Next>** to continue.

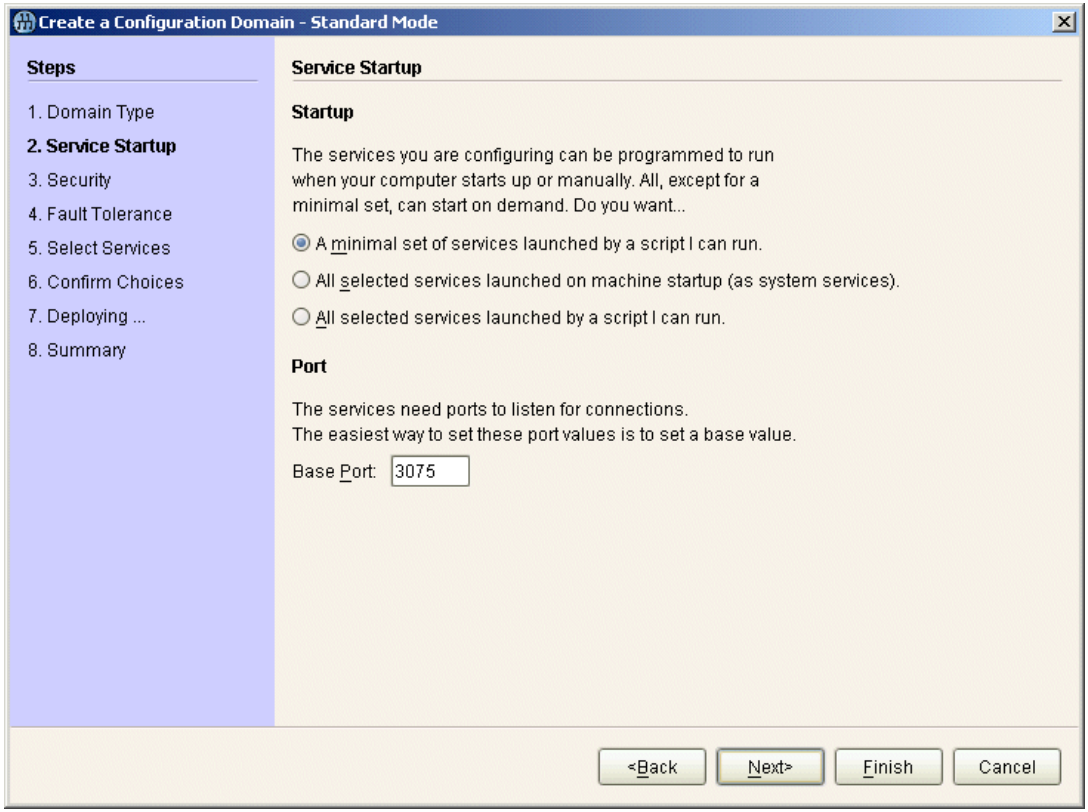

Figure 3: *The Service Startup Window*

Specify security settings

A **Security** window appears, as shown in Figure 4.

Under **Transports**, click the **Secure Communication (TLS/HTTPS)** radiobutton. Under **Security Features**, select the **IONA Security Service** option and the **Enable Access Control for Core Services** option.

Click **Next>** to continue.

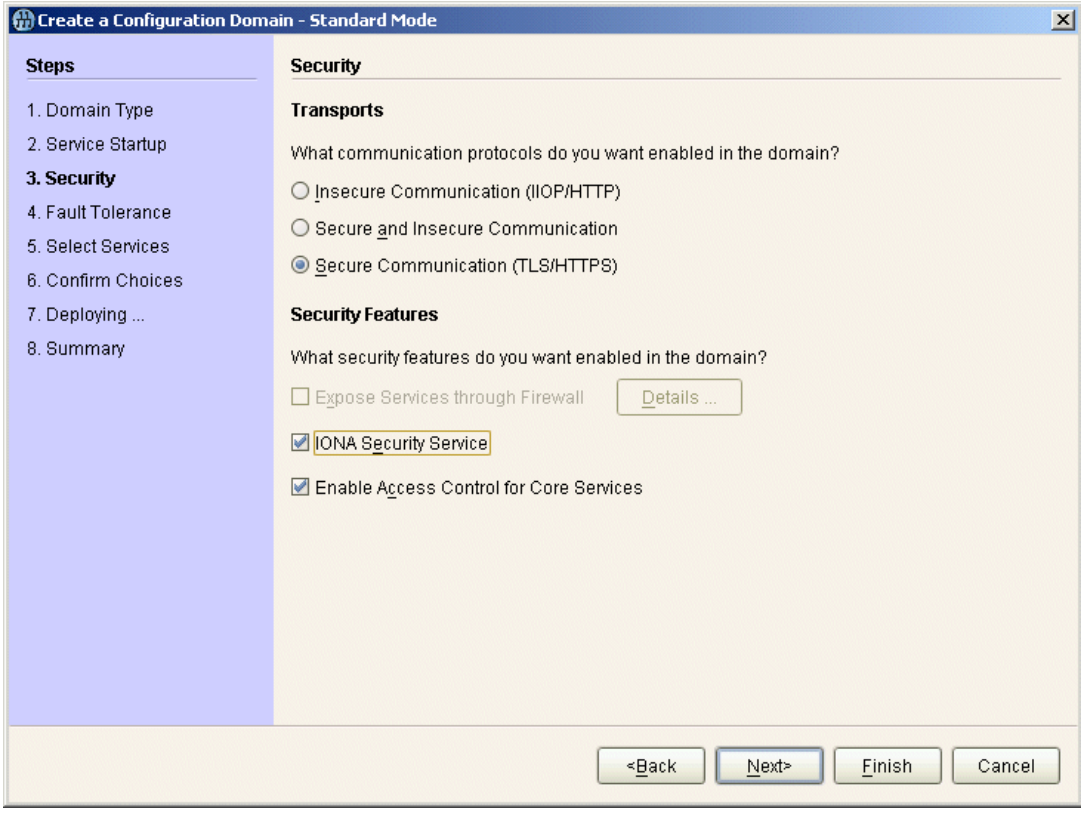

Figure 4: *The Security Window*

Specify fault tolerance settings

A **Fault Tolerance** window appears, as shown in Figure 5.

You can leave the settings in this Window at their defaults.

Click **Next>** to continue.

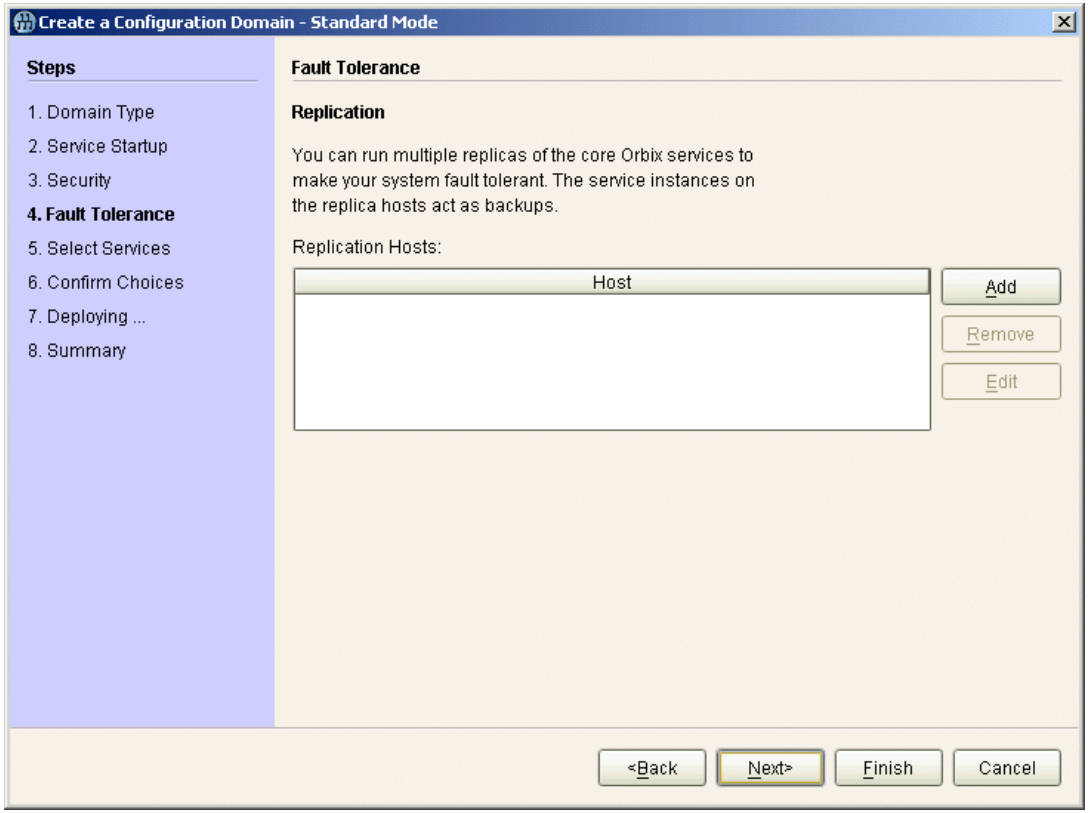

Figure 5: *The Fault Tolerance Window*

Select services

A **Select Services** window appears, as shown in Figure 6.

In the Select Services window, select the following services and components for inclusion in the configuration domain: **Location**, **Node daemon**, **Management**, **CORBA Interface Repository**, **CORBA Naming**, **IONA Security**, and **demos**.

Click **Next>** to continue.

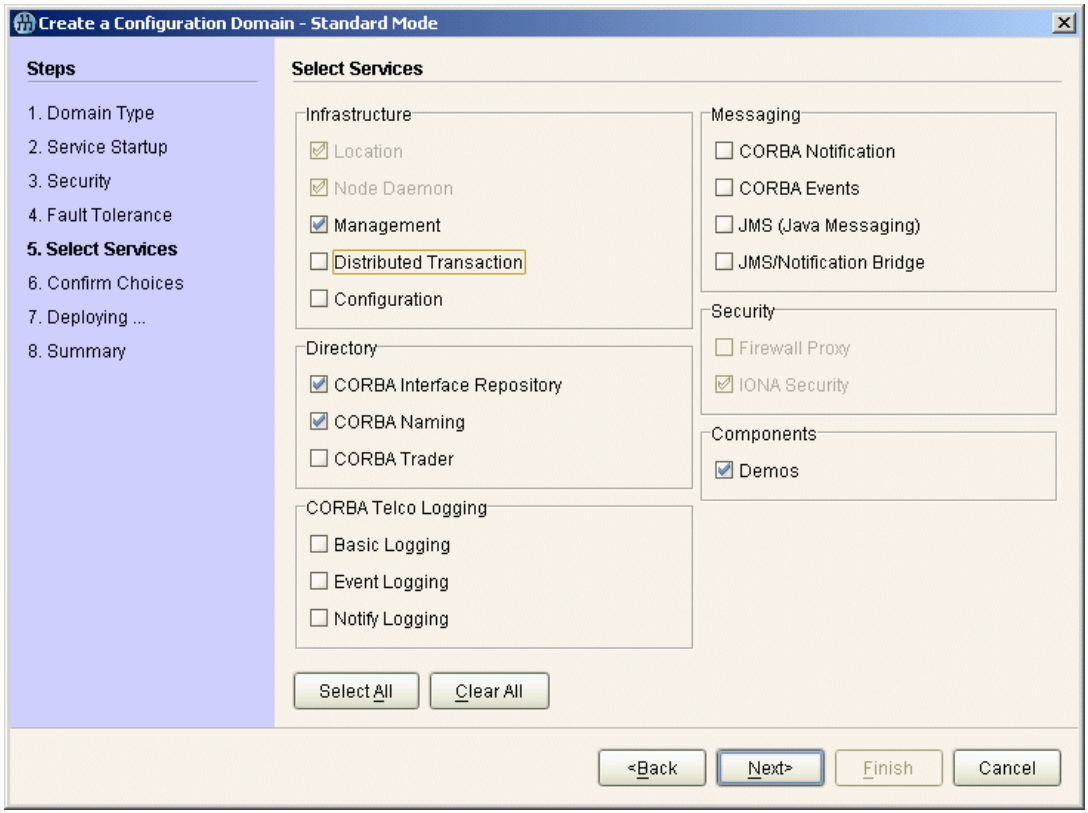

Figure 6: *The Select Services Window*

Confirm choices

You now have the opportunity to review the configuration settings in the **Confirm Choices** window, Figure 7. If necessary, you can use the **<Back** button to make corrections.

Click **Next>** to create the secure configuration domain and progress to the next window.

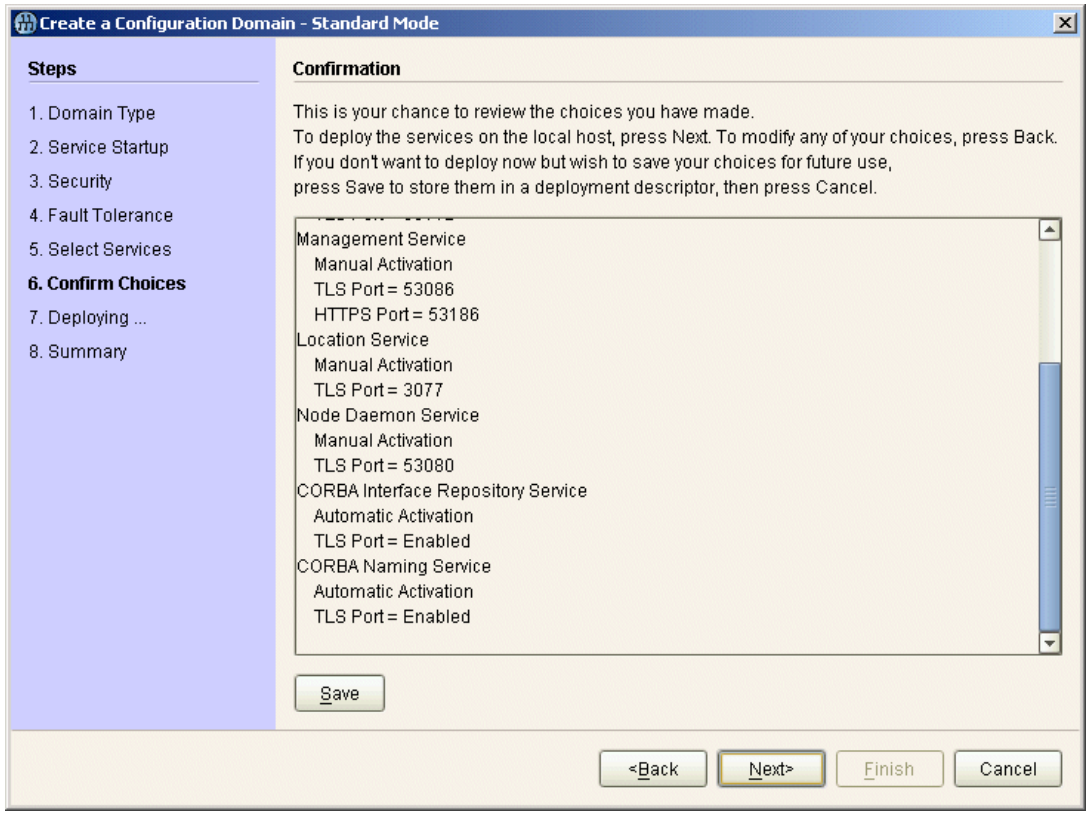

Figure 7: *The Confirm Choices Window*

Finish configuration

The `itconfigure` utility now creates and deploys the `secure` configuration domain, writing files into the *OrbixInstallDir*`/etc/bin`, *OrbixInstallDir*`/etc/domain`, *OrbixInstallDir*`/etc/log`, and *OrbixInstallDir*`/var` directories.

If the configuration domain is created successfully, you should see a **Summary** window with a message similar to that shown in Figure 8.

Click **Finish** to quit the `itconfigure` utility.

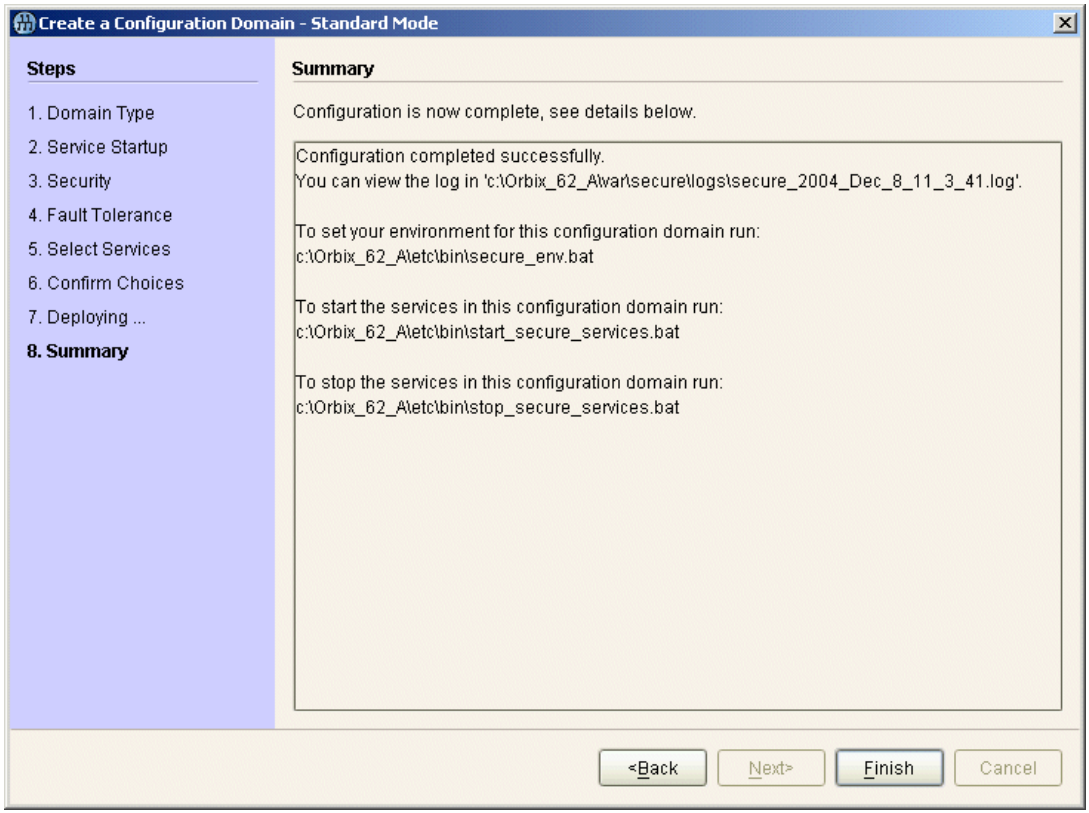

Figure 8: *Configuration Summary*

Running a Secure CORBA Demonstration

Overview

This section describes how to run the secure CORBA demonstration, which is a three-tier application that illustrates the SSL/TLS, username/password authentication, and identity assertion features.

Prerequisites

Before running this demonstration, you must have created a secure configuration domain—see "Creating a Secure Domain" on page 4.

Demonstration location

The secure CORBA demonstration is located in the following directory:

ASPInstallDir`/asp/`*Version*`/demos/common/is2`

Where *ASPInstallDir* is the directory where Orbix is installed.

Demonstration overview

Figure 9 gives an overview of the secure CORBA demonstration.

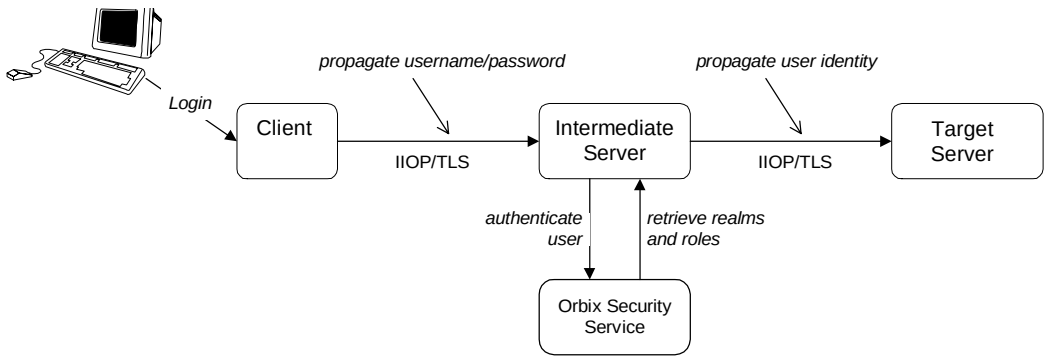

Figure 9: *CORBA Secure Demonstration Overview*

Steps

To build and run the secure CORBA demonstration, perform the following steps:

1. Build the demonstration.
2. Start the Orbix services.

3. Run the target server.
4. Run the intermediate server.
5. Run the client.

Build the demonstration

To build the demonstration, open a new command prompt and enter the following commands:

Windows

```
> ASPInstallDir\etc\bin\secure_env.bat
> cd ASPInstallDir\asp\Version\demos\common\is2
> itant
```

UNIX

```
% . ASPInstallDir/etc/bin/secure_env.sh
% cd ASPInstallDir/asp/Version/demos/common/is2
% itant
```

Start the Orbix services

To start the Orbix services, enter the following command at the command prompt:

Windows

```
> ASPInstallDir\etc\bin\start_secure_services.bat
```

UNIX

```
% ASPInstallDir/etc/bin/start_secure_services
```

Run the target server

To run the target server, open a new command prompt and enter the following commands:

Windows and J2SE (JDK) 1.3.x

```
> ASPInstallDir\etc\bin\secure_env.bat
> cd ASPInstallDir\asp\Version\demos\common\is2
> java -classpath .\java\classes;"%CLASSPATH%" is2.Server
```

Windows and J2SE (JDK) 1.4.x

```
> ASPInstallDir\etc\bin\secure_env.bat
> cd ASPInstallDir\asp\Version\demos\common\is2
> java -Djava_endorsed.dirs="ASPInstallDir\\lib\\art\\omg\\5"
   -classpath .\java\classes;"%CLASSPATH%" is2.Server
```

UNIX and J2SE (JDK) 1.3.x

```
% . ASPInstallDir/etc/bin/secure_env.sh
% cd ASPInstallDir/asp/Version/demos/common/is2
```

```
% java -classpath ./java/classes:$CLASSPATH is2.Server
```

UNIX and J2SE (JDK) 1.4.x

```
% . ASPInstallDir/etc/bin/secure_env.sh
% cd ASPInstallDir/asp/Version/demos/common/is2
% java -Djava_endorsed.dirs=ASPInstallDir/lib/art/omg/5 -classpath
   ./java/classes:$CLASSPATH is2.Server
```

Run the intermediate server

To run the intermediate server, open a new command prompt and enter the following commands:

Windows and J2SE (JDK) 1.3.x

```
> ASPInstallDir\etc\bin\secure_env.bat
> cd ASPInstallDir\asp\Version\demos\common\is2
> java -classpath .\java\classes;"%CLASSPATH%"
   is2.IntermediateServer
```

Windows and J2SE (JDK) 1.4.x

```
> ASPInstallDir\etc\bin\secure_env.bat
> cd ASPInstallDir\asp\Version\demos\common\is2
> java -Djava_endorsed.dirs="ASPInstallDir\\lib\\art\\omg\\5"
   -classpath .\java\classes;"%CLASSPATH%" is2.IntermediateServer
```

UNIX and J2SE (JDK) 1.3.x

```
% . ASPInstallDir/etc/bin/secure_env.sh
% cd ASPInstallDir/asp/Version/demos/common/is2
% java -classpath ./java/classes:$CLASSPATH is2.IntermediateServer
```

UNIX and J2SE (JDK) 1.4.x

```
% . ASPInstallDir/etc/bin/secure_env.sh
% cd ASPInstallDir/asp/Version/demos/common/is2
% java -Djava_endorsed.dirs=ASPInstallDir/lib/art/omg/5 -classpath
   ./java/classes:$CLASSPATH is2.IntermediateServer
```

Note: The intermediate server must run in the same directory as the target server.

Run the client

To run the client, open a new command prompt and enter the following commands:

Windows and J2SE (JDK) 1.3.x

```
> ASPInstallDir\etc\bin\secure_env.bat
> cd ASPInstallDir\asp\Version\demos\common\is2
```

```
> java -classpath .\java\classes;"%CLASSPATH%" is2.Client -user
   alice
```

Windows and J2SE (JDK) 1.4.x

```
> ASPInstallDir\etc\bin\secure_env.bat
> cd ASPInstallDir\asp\Version\demos\common\is2
> java -Djava_endorsed.dirs="ASPInstallDir\\lib\\art\\omg\\5"
   -classpath .\java\classes;"%CLASSPATH%" is2.Client -user alice
```

UNIX and J2SE (JDK) 1.3.x

```
% . ASPInstallDir/etc/bin/secure_env.sh
% cd ASPInstallDir/asp/Version/demos/common/is2
% java -classpath ./java/classes:$CLASSPATH is2.Client -user alice
```

UNIX and J2SE (JDK) 1.4.x

```
% . ASPInstallDir/etc/bin/secure_env.sh
% cd ASPInstallDir/asp/Version/demos/common/is2
% java -Djava_endorsed.dirs=ASPInstallDir/lib/art/omg/5 -classpath
   ./java/classes:$CLASSPATH is2.Client -user alice
```

Note: The client must run in the same directory as the target and intermediate servers.

Debugging with the openssl Utility

Overview

The `openssl` utility included with Orbix provides two powerful tools for debugging SSL/TLS client and server applications, as follows:

- `openssl s_client`—an SSL/TLS test client, which can be used to test secure Orbix servers. The test client can connect to a secure port, while providing a detailed log of the steps performed during the SSL/TLS handshake.
- `openssl s_server`—an SSL/TLS test server, which can be used to test secure Orbix clients. The test server can simulate a bare bones SSL/TLS server (handshake only). Additionally, by supplying the `-www` switch, the test server can also simulate a simple secure Web server.

References

For complete details of the `openssl s_client` and the `openssl s_server` commands, see the following OpenSSL documentation pages:

- http://www.openssl.org/docs/apps/s_client.html
- http://www.openssl.org/docs/apps/s_server.html

Debugging example

Consider the `is2` demonstration discussed in the previous section, "Running a Secure CORBA Demonstration" on page 13. This demonstration consists of a client, an intermediate server and a target server.

To demonstrate SSL debugging, you can use the `openssl` test client to connect directly to the target server.

Debugging steps

The following table shows the steps required to debug a secure server by connecting to that server using the `openssl` test client:

Step	Action
1	Convert the client certificate to PEM format.
2	Run the target server.
3	Obtain the target server's IP port.

Step	Action
4	Run the test client.

Convert the client certificate to PEM format

Certificates for Orbix applications are deployed in PKCS#12 format, whereas the `openssl` test client requires the certificate to be in PEM format (a format that is proprietary to OpenSSL). It is, therefore, necessary to convert the client certificate to the PEM format.

For example, given the certificate `admin.p12` (located in the *OrbixInstallDir*`/asp/Version/etc/tls/x509/certs/demos` directory), you can convert the certificate to PEM format as follows.

1. Run the `openssl pkcs12` command, as follows:

   ```
   openssl pkcs12 -in admin.p12 -out admin.pem
   ```

   When you run this command you are prompted to enter, first of all, the pass phrase for the `admin.p12` file and then to enter a pass phrase for the newly created `admin.pem` file.

2. The `admin.pem` file generated in the previous step contains a CA certificate, an application certificate, and the application certificate's private key. Before you can use the `admin.pem` file with the `openssl` test client, however, you must remove the CA certificate from the file. That is, the file should contain only the application certificate and its private key.

   For example, after deleting the CA certificate from the `admin.pem` file, the contents of the file should look something like the following:

```
Bag Attributes
    localKeyID: 6A F2 11 9B A4 69 16 3C 3B 08 32 87 A6 7D 7C 91
   C1 E1 FF 4A
    friendlyName: Administrator
subject=/C=US/ST=Massachusetts/O=ABigBank -- no warranty -- demo
   purposes/OU=Administration/CN=Administrator/emailAddress=admi
   nistrator@abigbank.com
issuer=/C=US/ST=Massachusetts/L=Boston/O=ABigBank -- no warranty
   -- demo purposes/OU=Demonstration Section -- no warranty
   --/CN=ABigBank Certificate
   Authority/emailAddress=info@abigbank.com
-----BEGIN CERTIFICATE-----
MIIEiTCCA/KgAwIBAgIBATANBgkqhkiG9w0BAQQFADCB5jELMAkGA1UEBhMCVVMx
FjAUBgNVBAgTDU1hc3NhY2h1c2V0dHMxDzANBgNVBAcTBkJvc3RvbjExMC8GA1UE
```

```
subject=/C=US/ST=Massachusetts/O=ABigBank -- no warranty -- demo
   purposes/OU=Administration/CN=Administrator/emailAddress=admi
   nistrator@abigbank.com
issuer=/C=US/ST=Massachusetts/L=Boston/O=ABigBank -- no warranty
   -- demo purposes/OU=Demonstration Section -- no warranty
   --/CN=ABigBank Certificate
   Authority/emailAddress=info@abigbank.com
-----BEGIN CERTIFICATE-----
MIIEiTCCA/KgAwIBAgIBATANBgkqhkiG9w0BAQQFADCB5jELMAkGA1UEBhMCVVMx
FjAUBgNVBAgTDU1hc3NhY2h1c2V0dHMxDzANBgNVBAcTBkJvc3RvbjExMC8GA1UE
```

```
ZgNtAB+XF9vrA5XZHNsU6RBeXMVSrUlOGzdVrCnojd6d8Be7Q7KBSHDV9XzZlPKp
7DYVn5DyFSEQ7kYs9dsaZ5Id5iNkMJiscPp7AL2SJAWpYlUfEN5gFnIYiwXP1ckF
STTiq+BG8UPPm6G3KGgRZMZ0Ih7DySZufbE24NIrN74kXV9Vf/RpxzNiMz/PbLdG
6wiyp47We/4OqxLv8YIjGGEdYyaB/Y7XEyE9ZL74Dc3CcuSvtA2fC8hU3cXjKBu7
YsVz/Dq8G0w223owpZ0Qz2KUl9CLq/hmYLOJt1yLVoaGZuJ1CWXdgX0dComDOR8K
aIaUagy/Gz2zys20N5WRK+s+HzqoB0vneOy4Z1Ss71HfGAUemiRTAI8DXizgyHYK
5m6iSSB961xOM7YI58JYOGNLMXzlLmCUAyCQhklWGJFEN4cZBrkh5o6r+U4FcwhF
dvDoBu39Xie5gHFrJU86qhzxi202h0sO2vexvujSGyNy009PJGkEAhJGfOG+a2Qq
VBwuUZqo0zIJ6gUrMV1LOAWwL7zFxyKaF5lijF1C9KxtEKm0393zag==
-----END RSA PRIVATE KEY-----
```

Run the target server

Run the target server, as described in "Running a Secure CORBA Demonstration" on page 13.

Obtain the target server's IP port

In this demonstration, the server writes an IOR file, `target_server.ior`, to the *OrbixInstallDir*`/asp/`*Version*`/demos/common/is2` directory as it starts up. You can extract the target server's IP port from this IOR file using the `iordump` utility.

From a command prompt, use the `iordump` utility to parse the `target_server.ior` file as follows:

```
iordump target_server.ior
```

This dumps the parsed contents of the IOR to the console window. The relevant portion of the output is the `SSL_SEC_TRANS` tagged component, which looks something like the following:

```
                Component 1:
>> +108 [00][00][00][14]
                          Tag: (20) SSL_SEC_TRANS
>> +112 [00][00][00][08]
                          Component length: 8 bytes
>> +116 [00]
                          Component Byte Order: (0) Big Endian
>> +117 [00]
       (padding)
>> +118 [00][7e]
                       Target supports: Integrity Confidentiality
   DetectReplay DetectMisordering EstablishTrustInTarget
   EstablishTrustInClient
>> +120 [00][5e]
                       Target requires: Integrity Confidentiality
   DetectReplay DetectMisordering EstablishTrustInClient
```

```
>> +122 [0b][8b]
                          SSL port: 2955
```

In this example, the target server's IP port is 2955.

Run the test client

To run the openssl test client, open a command prompt, change directory to the directory containing the admin.pem file, and enter the following command:

```
openssl s_client -connect localhost:2955 -ssl3 -cert admin.pem
```

When you enter the command, you are prompted to enter the pass phrase for the admin.pem file.

The openssl s_client command switches can be explained as follows:

-connect *host*:*port*
: Open a secure connection to the specified *host* and *port*.

-ssl3
: This option configures the client to initiate the handshake using SSL v3 (the default is SSL v2). To see which SSL version (or versions) the target server is configured to use, check the value of the policies:mechanism_policy:protocol_version variable in the Orbix configuration file. Orbix servers can also be configured to use TLS v1, for which the corresponding openssl command switch is -tls1.

-cert admin.pem
: Specifies admin.pem as the test client's own certificate. The PEM file should contain only application certificate and the application certificate's private key. The PEM file should *not* contain a complete certificate chain.

  If your server is not configured to require a client certificate, you can omit the -cert switch.

Other command switches
: The openssl s_client command supports numerous other command switches, details of which can be found on the OpenSSL document pages (see "References" on page 17). Two of the more interesting switches are -state and -debug, which log extra details to the command console during the handshake.

Where do I go from here?

Overview

To help you get started in the wide-ranging field of security, you might find it helpful to focus on one of the following fundamental tasks:

- I want to customize the sample domain to make it fully secure.
- I want to security-enable a CORBA application.
- I want to write a security-aware CORBA application.
- I want to integrate a third-party enterprise security system.
- I want to replace the default SSL/TLS toolkit.

I want to customize the sample domain to make it fully secure

The sample configuration domains generated by the `itconfigure` utility are *not* fully secure, because the X.509 certificates used by the Orbix services are insecure demonstration certificates. To perform basic customization of a secure configuration domain, see the following reference:

- "Securing Orbix Services" on page 209.

I want to security-enable a CORBA application

To security-enable a CORBA application, see the following reference:

- "Securing CORBA Applications" on page 63.

I want to write a security-aware CORBA application

To write a security-aware CORBA application, see the following references:

- "Programming Policies" on page 447.
- "Authentication" on page 461.
- "Validating Certificates" on page 499.

I want to integrate a third-party enterprise security system

The Orbix Security Framework provides a facility for integrating with third-part enterprise security systems, such as LDAP, through a pluggable system of security adapters. For details of how this works, see the following reference:

- "Configuring the Orbix Security Service" on page 141.

For details of how to write your own custom adapter, see the following reference:

- "Developing an iSF Adapter" on page 519.

I want to replace the default SSL/TLS toolkit

By default, Orbix uses the SSL/TLS toolkit from Baltimore Technologies. IONA's SSL/TLS toolkit replaceability feature enables you to replace the underlying SSL/TLS toolkit used by an Orbix applications. For details, see the following chapter:

- "Choosing an SSL/TLS Toolkit" on page 269.

CHAPTER 2

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 10 shows an example of an iSF system that features a standalone Orbix security service, which can service remote requests for security-related functions.

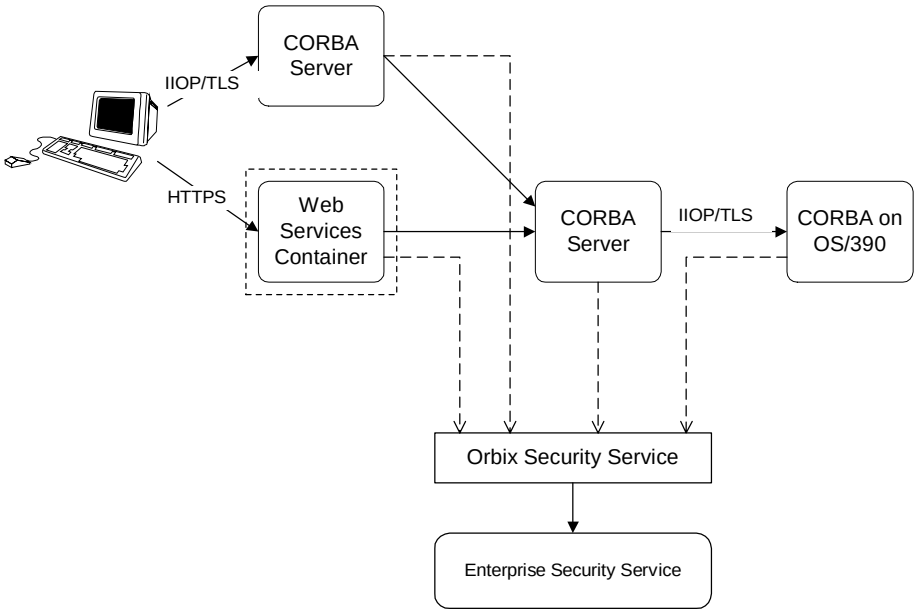

Figure 10: *Example System with a Standalone Orbix Security Service*

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 34.

Propagating security credentials

The example in Figure 10 on page 28 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).

Optional iSF components

The Orbix security service includes the following optional components that can be enabled to provide additional security features:

- Single sign-on.

Single sign-on

Single sign-on means that once an application has authenticated a particular user, it is relatively easy for other secure applications to access that user's security data.

When single sign-on is enabled, the Orbix security service creates an association between an SSO token and a user session. Any application that has the user's SSO token can then use it to access the user's session data.

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

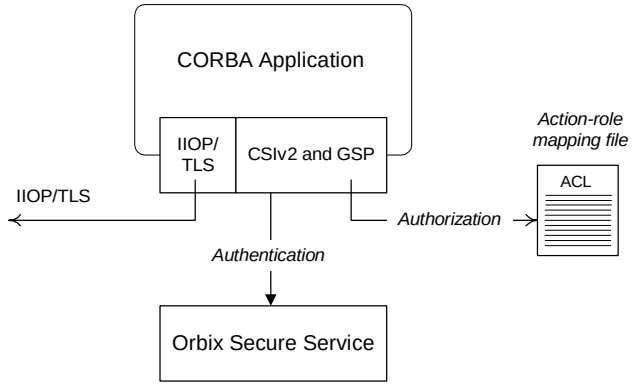

Figure 11: *Security Plug-Ins in a CORBA Application*

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" on page 543.

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 3

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 (C++ and Java). 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.

SSL/TLS toolkit replaceability

Orbix has an SSL/TLS toolkit replaceability feature that enables you to replace completely the underlying toolkit that implements SSL/TLS in Orbix. Currently, you have a choice between the Baltimore toolkit (all platforms) and the Schannel toolkit (Windows only).

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.

Obtaining Credentials from X.509 Certificates

Obtaining own credentials

This section discusses how an application's own credentials are initially obtained from an X.509 certificate. An application's own credentials are the credentials that the application normally uses to identify itself to other applications.

Comparison of PKCS#12 and PKCS#11

Two mechanisms for obtaining own credentials are described in this section:

- PKCS#12—credentials obtained from a PKCS#12 file.
- PKCS#11—credentials obtained from a smart card. Orbix uses the PKCS#11 interface to communicate with the smart card.

In this section

This section contains the following subsections:

Obtaining Certificate Credentials from a File

Creating credentials using the principal sponsor

The simplest way for a client to obtain certificate credentials is to configure an *SSL/TLS principal sponsor* for the client application. This principal sponsor can be initialized by editing the Orbix configuration—see "Specifying an Application's Own Certificate" on page 363.

Creating credentials from a PKCS#12 file

Figure 12 illustrates how the principal sponsor creates credentials from a PKCS#12 file.

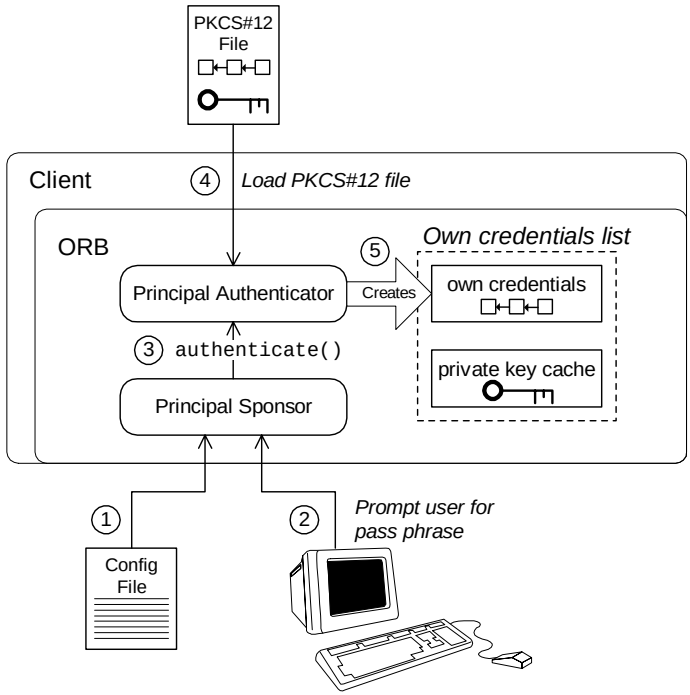

Figure 12: *Creating Credentials for a Client Application Using PKCS#12*

Steps for creating credentials

The principal sponsor automates the steps to create credentials, as follows:

1. The principal sponsor reads the client configuration file to discover which authentication method to use.
2. If the authentication method is PKCS#12, the principal sponsor obtains the pass phrase to decrypt the client's certificate chain and private key. The pass phrase is obtained either by running a login utility that prompts the user for the pass phrase, or by reading the client configuration file—see "Providing a Certificate Pass Phrase" on page 368.
3. The principal sponsor requests the *principal authenticator* to generate credentials for the client by invoking the `authenticate()` operation, passing the following data as parameters:
   - Pass phrase,
   - PKCS#12 file name.
4. The principal authenticator loads the PKCS#12 file to obtain the client identity. The PKCS#12 file contains an encrypted *X.509 certificate chain* and an encrypted *private key*.
5. If the authentication step is successful, the principal authenticator creates an *own credentials* object, of `SecurityLevel2::Credentials` type. The own credentials object is cached in memory along with its private key.

How PKCS#12 credentials are used in an SSL/TLS handshake

Figure 13 illustrates how PKCS#12 credentials are used during an SSL/TLS handshake, showing only the portion of the handshake where the server verifies the client's identity.

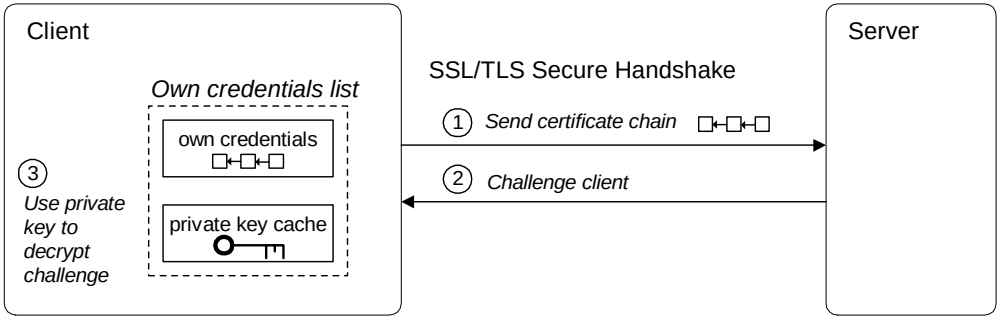

Figure 13: *Using PKCS#12 Credentials to Authenticate a Client to a Server*

PKCS#12 handshake steps

During an SSL/TLS handshake, the client authenticates itself to the server as follows:

1. At a certain point during the SSL/TLS handshake, the client sends an X.509 certificate chain (which has been cached in an own credentials object) to the server.
2. The server sends a challenge message, encrypted using the client's public key.
3. The client uses the private key (cached in memory) to decrypt the challenge message.
4. Having successfully answered the server challenge, the client proceeds to the next stage of the handshake (not shown).

Obtaining Certificate Credentials from a Smart Card

Creating credentials using the PKCS#11 interface

Figure 14 illustrates how the SSL/TLS principal sponsor creates certificate credentials using the PKCS#11 interface—see "Specifying an Application's Own Certificate" on page 363.

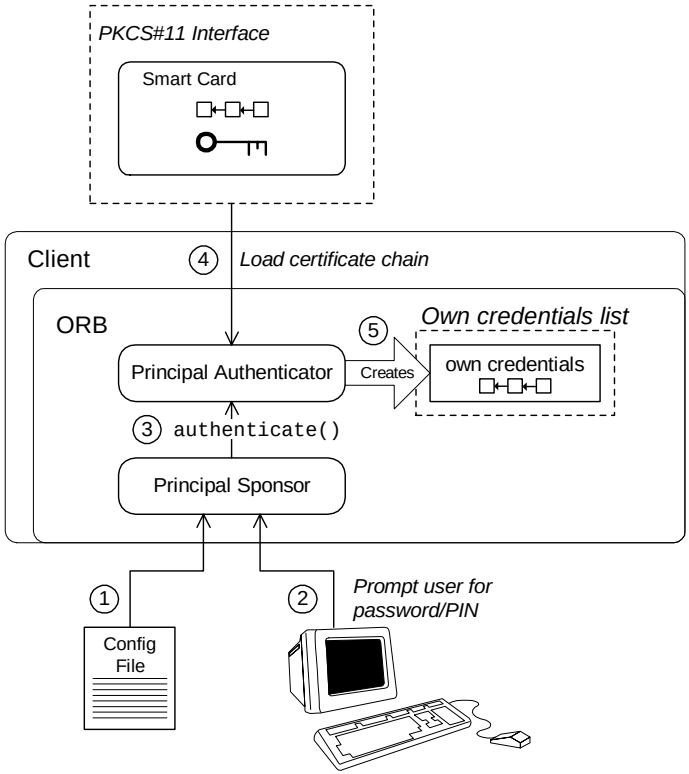

Figure 14: *Creating Credentials for a Client Application Using PKCS#11*

Steps for creating credentials

The principal sponsor automates the steps to create credentials, as follows:

1. The principal sponsor reads the client configuration file to discover which authentication method to use.
2. If the authentication method is PKCS#11, the principal sponsor obtains the smart card's PIN to gain access to the smart card. The PIN is obtained either by running a login utility that prompts the user for the PIN, or by reading the client configuration file—see "Providing a Smart Card PIN" on page 372.
3. The principal sponsor requests the *principal authenticator* to generate credentials for the client by invoking the `authenticate()` operation, passing the following data:
   - Provider name,
   - Slot number,
   - PIN or pass phrase.
4. The principal authenticator communicates with the smart card using the PKCS#11 interface to obtain the client identity. The principal authenticator uploads *only* the X.509 certificate chain. The private key is left on the smart card.
5. If the authentication step is successful, the principal authenticator creates an *own credentials* object, of `SecurityLevel2::Credentials` type. The own credentials object is cached in memory *but its private key is not stored in memory*.

How PKCS#11 credentials are used in an SSL/TLS handshake

Figure 15 illustrates how PKCS#11 credentials are used during an SSL/TLS handshake, showing only the portion of the handshake where the server verifies the client's identity.

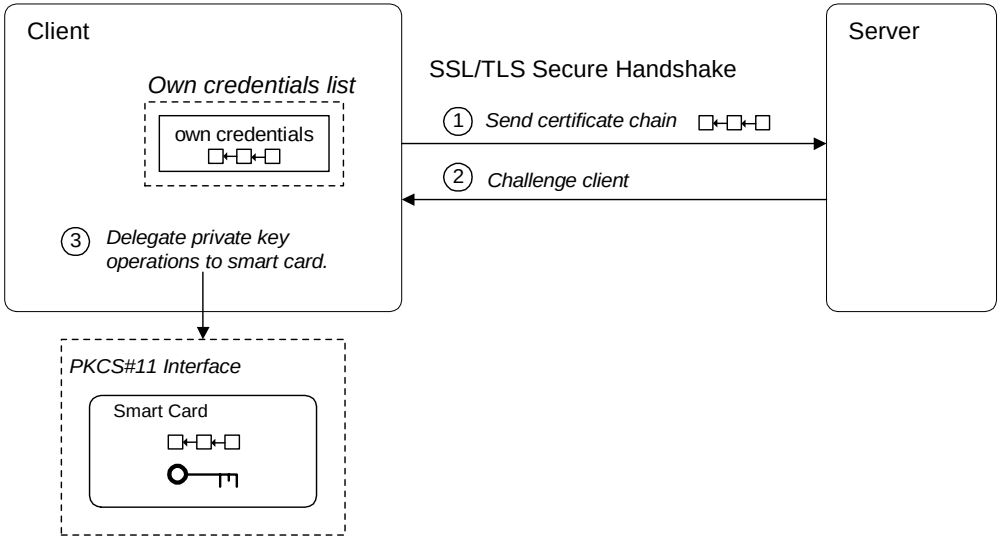

Figure 15: *Using PKCS#11 Credentials to Authenticate a Client to a Server*

PKCS#11 handshake steps

During an SSL/TLS handshake, the client authenticates itself to the server as follows:

1. At a certain point during the SSL/TLS handshake, the client sends an X.509 certificate chain (which has been cached in an own credentials object) to the server.
2. The server sends a challenge message, encrypted using the client's public key.

3. The client delegates the challenge message to the smart card, using the PKCS#11 interface. The smart card uses the appropriate private key to decrypt the challenge message. Because the smart card has a built-in processor, it is able to perform the private key calculations in place. The private key never leaves the smart card.
4. Having successfully answered the server challenge, the client proceeds to the next stage of the handshake (not shown).

Note: At no point during the handshake is the smart card's private key loaded into memory.

CHAPTER 4

Securing CORBA Applications

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

In this chapter

This chapter discusses the following topics:

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

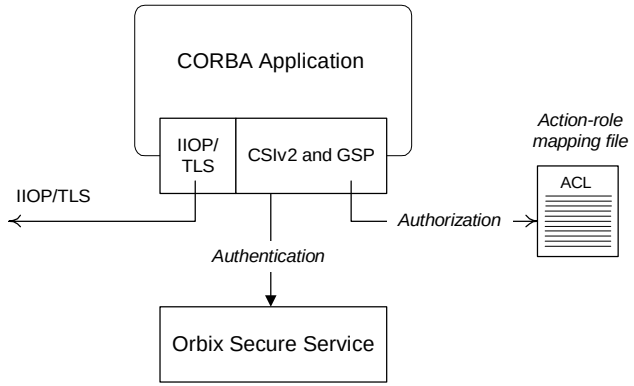

Figure 16: *A Secure CORBA Application within the iSF*

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
- 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 66 for details on how to enable IIOP/TLS in a CORBA application.

CSIv2 plug-in

The CSIv2 plug-in, `csi`, provides a client authentication mechanism for CORBA applications. The authentication mechanism is based on a username and a password. When the CSIv2 plug-in is configured for use with the iSF, the username and password are forwarded to a central Orbix security service 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, `gsp`, provides authorization by checking a user's roles against the permissions stored in an action-role mapping file. 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 (generated by the `itconfigure` utility with security enabled—see "Creating a Secure Domain" on page 4).

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

If a location domain, *DomainName*, is generated with security enabled and demonstration configurations enabled, the domain will include several sample configurations that can be used as templates for configuring SSL/TLS. Within the default domain configuration (either in the *DomainName*`.cfg` file or in the CFR service), you can find the following sample SSL/TLS configuration scopes:

- `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.
`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:trusted_ca_list_policy =
    "ASPInstallDir\asp\6.0\etc\tls\x509\trusted_ca_lists\ca_list1.
    pem";

4   policies:mechanism_policy:protocol_version = "SSL_V3";
    policies:mechanism_policy:ciphersuites =
    ["RSA_WITH_RC4_128_SHA", "RSA_WITH_RC4_128_MD5"];

5   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')
6       principal_sponsor:use_principal_sponsor = "false";

7       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 before the `iiop_tls` plug-in in the ORB plug-ins list.

   If you plan to use the full Orbix Security Framework, you should include the `gsp` plug-in in the ORB plug-ins list as well—see "Securing Two-Tier CORBA Systems with CSI" on page 78.

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

   If you plan to use the full Orbix Security Framework, you should use the `binding:client_binding_list` as shown in "Client configuration" on page 79 instead.

3. An SSL/TLS application needs a list of trusted CA certificates, which it uses to determine whether or not to trust certificates received from other SSL/TLS applications. You must, therefore, edit the `policies:trusted_ca_list_policy` variable to point at a list of trusted certificate authority (CA) certificates. See "Specifying Trusted CA Certificates" on page 361.

   Note: If using Schannel as the underlying SSL/TLS toolkit (Windows only), the `policies:trusted_ca_list_policy` variable is ignored. Within Schannel, the trusted root CA certificates are obtained from the Windows certificate store.

4. The SSL/TLS mechanism policy specifies the default security protocol version and the available cipher suites—see "Specifying Cipher Suites" on page 343.

5. 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.
6. 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`.
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 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')
    ...
```

Example 2: *Sample SSL/TLS Server Configuration*

```
    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 = "pkcs12_file";
5       principal_sponsor:auth_method_data =
   ["filename=ASPInstallDir\asp\6.0\etc\tls\x509\certs\demos\bank
   _server.p12"];

        # Specific SSL/TLS client configuration settings
        # (copied from 'demos.tls.secure_client_with_cert')
6       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 70
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 a PKCS#12 file. For alternative methods, see "Specifying an Application's Own Certificate" on page 363.

   Note: If using Schannel as the underlying SSL/TLS toolkit (Windows only), the `principal_sponsor:auth_method_id` value must be `security_label` instead of `pkcs12_file`.

5. Replace the X.509 certificate, by editing the `filename` option in the `principal_sponsor:auth_method_data` configuration variable to point at a custom X.509 certificate. The `filename` value should be initialized with the location of a certificate file in PKCS#12 format—see "Specifying an Application's Own Certificate" on page 363 for more details.

   Note: If using Schannel as the underlying SSL/TLS toolkit (Windows only), you would set the `label` option instead of the `filename` option in the `principal_sponsor:auth_method_data` configuration variable. The `label` specifies the common name (CN) from the application certificate's subject DN.

   For details of how to specify the certificate's pass phrase, see "Providing a Pass Phrase or PIN" on page 367.
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 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 72. 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 327.
- "Configuring SSL/TLS Authentication" on page 353.

Key distribution management

It is possible to configure your CORBA server so that the certificate pass phrase is supplied automatically by the key distribution management (KDM) service. For details, see the following reference:

- "Automatic Activation of Secure Servers" on page 381.

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 CSI

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.

Two-tier CORBA system

Figure 17 shows a basic two-tier CORBA system in the iSF, featuring a client and a target server.

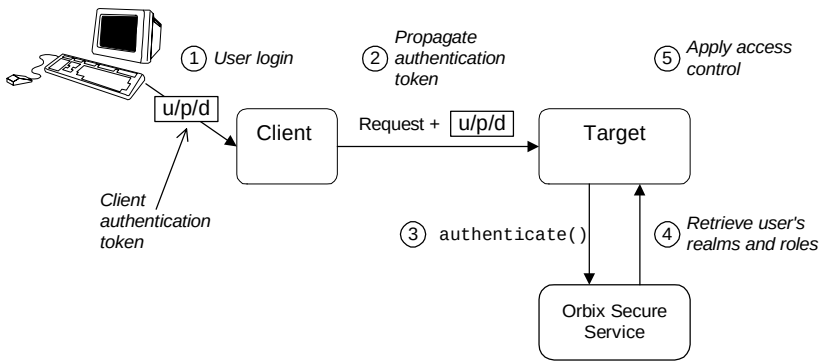

Figure 17: *Two-Tier CORBA System in the iSF*

Scenario description

The scenario shown in Figure 17 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 can either be an empty string (acts as a wildcard) or must match the value of the `policies:csi:auth_over_transport:server_domain_name` configuration variable set on the server side.
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 Orbix security service.
4	If authentication is successful, the Orbix security service returns the user's realms and roles.
5	The iSF controls access to the target's IDL interfaces by consulting an *action-role mapping file* to determine what the user is allowed to do.

Client configuration

The CORBA client from Example 17 on page 78 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"];
```

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

```
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 {
5           # Specific SSL/TLS configuration settings.
            ...
            # Specific iSF configuration settings.
6           plugins:csi:allow_csi_reply_without_service_context =
        "false";
7           policies:csi:auth_over_transport:client_supports =
        ["EstablishTrustInClient"];

8           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 66 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 66.

6. This setting enforces strict checking of reply messages from the server, to make sure the server actually supports CSIv2.
7. This configuration setting specifies that the client supports sending username/password authentication data to a server.
8. 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.

   For more details on the CSI principal sponsor, see "Providing a Username and Password" on page 420.

Target configuration

The CORBA target server from Figure 17 on page 78 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";

5       plugins:gsp:authorization_realm = "AuthzRealm";
6       plugins:gsp:action_role_mapping_file = "ActionRoleURL";
```

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

```
7           # 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 = [];
        };
};
```

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 66.
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. The domain name embedded in a received CSIv2 credential must match the value of the `server_domain_name` variable on the server side or could be an empty string (acts as a wildcard).
5. This configuration setting specifies the iSF authorization realm, *AuthzRealm*, to which this server belongs. For more details about iSF authorization realms, see "iSF Authorization Realms" on page 177.
6. The `action_role_mapping` configuration variable specifies the location of an action-role mapping that controls access to the IDL interfaces implemented by the server. The file location is specified in an URL format, for example:
   `file:///security_admin/action_role_mapping.xml` (UNIX) or
   `file:///c:/security_admin/action_role_mapping.xml` (Windows).
   For more details about the action-role mapping file, see "CORBA Action-Role Mapping ACL" on page 194.

7. 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.

Related administration tasks

After securing your CORBA applications with iSF, you might need to perform related administration tasks, for example:

- See "Managing Users, Roles and Domains" on page 173.
- See "CORBA Action-Role Mapping ACL" on page 194.

Securing Three-Tier CORBA Systems with CSI

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.

Three-tier CORBA system

Figure 18 shows a basic three-tier CORBA system in the iSF, featuring a client, an intermediate server and a target server.

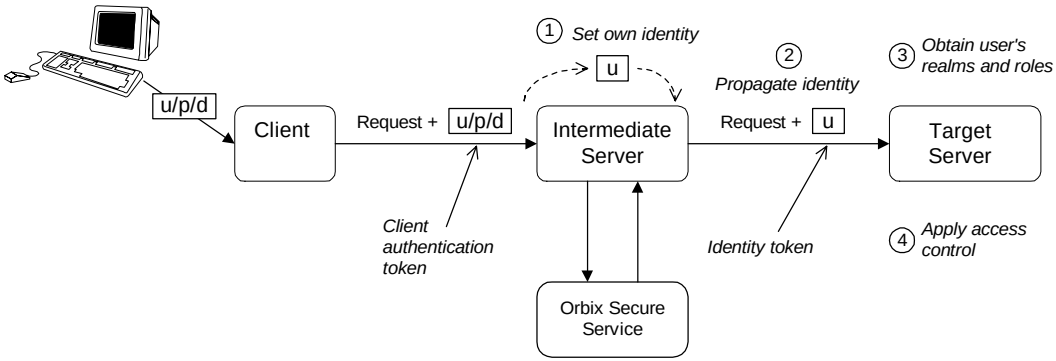

Figure 18: *Three-Tier CORBA System in the iSF*

Scenario description

The second stage of the scenario shown in Figure 18 (intermediate server invokes an operation on the target server) can be described as follows:

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

Client configuration

The client configuration for the three-tier scenario is identical to that of the two-tier scenario, as shown in "Client configuration" on page 79.

Intermediate configuration

The CORBA intermediate server from Figure 18 on page 84 can be configured as shown in Example 5.

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

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

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

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

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

7       plugins:gsp:authorization_realm = "AuthzRealm";
8       plugins:gsp:action_role_mapping_file = "ActionRoleURL";

9       # 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 = [];
    };
};
```

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 66.
2. This setting enforces strict checking of reply messages from the target, to make sure the target actually supports CSIv2.
3. 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.

4. This configuration setting specifies that the intermediate server *supports* receiving username/password authentication data from the client.
5. This configuration setting specifies that the intermediate server *requires* the client to send username/password authentication data.
6. The `server_domain_name` configuration variable sets the server's CSIv2 authentication domain name. The domain name embedded in a received CSIv2 credential must match the value of the `server_domain_name` variable on the server side or could be an empty string (acts as a wildcard).
7. This configuration setting specifies the iSF authorization realm, *AuthzRealm*, to which this server belongs. For more details about iSF authorization realms, see "iSF Authorization Realms" on page 177.
8. This configuration setting specifies the location of an action-role mapping that controls access to the IDL interfaces implemented by the server. The file location is specified in an URL format, for example: `file:///security_admin/action_role_mapping.xml` (UNIX) or `file:///c:/security_admin/action_role_mapping.xml` (Windows).

   For more details about the action-role mapping file, see "CORBA Action-Role Mapping ACL" on page 194.
9. 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.

Target configuration

The CORBA target server from Figure 18 on page 84 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.
```

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

```
    ...
    # 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"];

5       plugins:gsp:authorization_realm = "AuthzRealm";
6       plugins:gsp:action_role_mapping_file = "ActionRoleURL";

7       # 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 = [];
    };
};
```

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 66.
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 376.

   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. This configuration setting specifies the iSF authorization realm, *AuthzRealm*, to which this server belongs. For more details about iSF authorization realms, see "iSF Authorization Realms" on page 177.
6. This configuration setting specifies the location of an action-role mapping that controls access to the IDL interfaces implemented by the server. The file location is specified in an URL format, for example: `file:///security_admin/action_role_mapping.xml`.

   For more details about the action-role mapping file, see "CORBA Action-Role Mapping ACL" on page 194.
7. 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.

Related administration tasks

After securing your CORBA applications with iSF, you might need to perform related administration tasks, for example:

- See “Managing Users, Roles and Domains” on page 173.
- See “CORBA Action-Role Mapping ACL” on page 194.

X.509 Certificate-Based Authentication

Overview

This section describes how to enable X.509 certificate authentication with the iSF, based on a simple two-tier client/server scenario. In this scenario, the Orbix security service authenticates the client's certificate and retrieves roles and realms based on the identity of the certificate subject. When iSF certificate-based authentication is enabled, the X.509 certificate is effectively authenticated twice, as follows:

- *SSL/TLS-level authentication*—this authentication step occurs during the SSL/TLS handshake and is governed by Orbix configuration settings and programmable SSL/TLS policies.
- *iSF-level authentication and authorization*—this authentication step occurs after the SSL/TLS handshake and is performed by the Orbix security service working in tandem with the gsp plug-in.

Certificate-based authentication scenario

Figure 19 shows an example of a two-tier system, where authentication of the client's X.509 certificate is integrated with iSF.

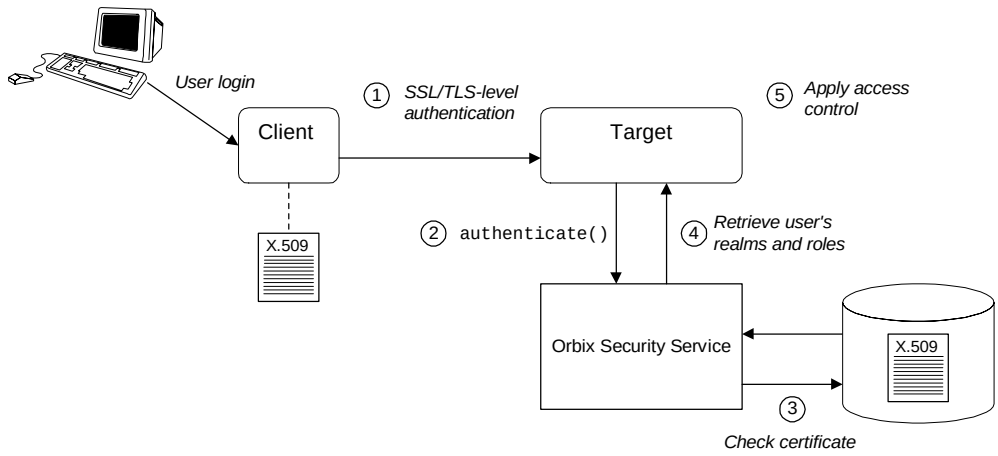

Figure 19: *Overview of iSF Certificate-Based Authentication*

Scenario description

The scenario shown in Figure 19 can be described as follows:

Stage	Description
1	When the client opens a connection to the server, the client sends its X.509 certificate as part of the SSL/TLS handshake. The server then performs SSL/TLS-level authentication, checking the certificate as follows: • The certificate is checked against the server's *trusted CA list* to ensure that it is signed by a trusted certification authority. • If a certificate constraints policy is set, the certificate is checked to make sure it satisfies the specified constraints. • If a certificate validator policy is set (by programming), the certificate is also checked by this policy.
2	The server then performs iSF-level authentication by calling `authenticate()` on the Orbix security service, passing the client's X.509 certificate as the argument.
3	The Orbix security service authenticates the client's X.509 certificate by checking it against a cached copy of the certificate. The type of checking performed depends on the particular *third-party enterprise security service* that is plugged into the Orbix security service.
4	If authentication is successful, the Orbix security service returns the user's realms and roles.
5	The iSF controls access to the target's IDL interfaces by consulting an *action-role mapping file* to determine what the user is allowed to do.

Client configuration

Example 7 shows a sample client configuration that you can use for the iSF certificate-based authentication scenario (Figure 19 on page 91).

Example 7: *Client Configuration for iSF Certificate-Based Authentication*

```
# Orbix Configuration File
corba_cert_auth
{
     orb_plugins = ["local_log_stream", "iiop_profile", "giop",
   "iiop_tls", "gsp"];

     event_log:filters = ["IT_GSP=*", "IT_CSI=*", "IT_TLS=*",
   "IT_IIOP_TLS=*", "IT_ATLI2_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", "GIOP+IIOP", "GIOP+IIOP_TLS"];

     client_x509
     {

   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=W:\art\etc\tls\x509\certs\demos\bob.p12",
   "password=bobpass"];
     };
};
```

The preceding client configuration is a typical SSL/TLS configuration. The only noteworthy feature is that the client must have an associated X.509 certificate. Hence, the `principal_sponsor` settings are initialized with the location of an X.509 certificate (provided in the form of a PKCS#12 file).

For a discussion of these client SSL/TLS settings, see "Sample client configuration" on page 70 and "Deploying Application Certificates" on page 303.

Target configuration

Example 8 shows a sample server configuration that you can use for the iSF certificate-based authentication scenario (Figure 19 on page 91).

Example 8: *Server Configuration for iSF Certificate-Based Authentication*

```
# Orbix Configuration File
corba_cert_auth
{
     orb_plugins = ["local_log_stream", "iiop_profile", "giop",
   "iiop_tls", "gsp"];

     event_log:filters = ["IT_GSP=*", "IT_CSI=*", "IT_TLS=*",
   "IT_IIOP_TLS=*", "IT_ATLI2_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", "GIOP+IIOP", "GIOP+IIOP_TLS"];

     server
     {
        policies:csi:auth_over_transport:authentication_service
   = "com.iona.corba.security.csi.AuthenticationService";

          principal_sponsor:use_principal_sponsor = "true";
          principal_sponsor:auth_method_id = "pkcs12_file";
1         principal_sponsor:auth_method_data =
   ["filename=OrbixInstallDir\etc\tls\x509\certs\demos\bank_server
   .p12", "password=bankserverpass"];

          binding:server_binding_list = ["CSI+GSP", "CSI",
   "GSP"];

          initial_references:IS2Authorization:plugin =
   "it_is2_authorization";

          plugins:it_is2_authorization:ClassName =
   "com.iona.corba.security.authorization.IS2AuthorizationPlugIn
   ";
```

Example 8: *Server Configuration for iSF Certificate-Based Authentication*

```
2          plugins:gsp:action_role_mapping_file =
       "file://W:\art\etc\tls\x509\..\..\..\..\art_svcs\etc\actionro
       lemapping_with_interfaces.xml";

           auth_x509
           {
3
       plugins:gsp:enable_security_service_cert_authentication =
       "true";


       policies:iiop_tls:target_secure_invocation_policy:supports =
       ["Integrity", "Confidentiality", "DetectReplay",
       "DetectMisordering", "EstablishTrustInTarget",
       "EstablishTrustInClient"];
4
       policies:iiop_tls:target_secure_invocation_policy:requires =
       ["Integrity", "Confidentiality", "DetectReplay",
       "DetectMisordering", "EstablishTrustInClient"];
           };
       };
};
```

The preceding server configuration can be explained as follows:

1. As is normal for an SSL/TLS server, you must provide the server with its own certificate. The simplest way to do this is to specify the location of a PKCS#12 file using the principal sponsor.
2. This configuration setting specifies the location of an action-role mapping file, which controls access to the server's interfaces and operations.
3. The `plugins:gsp:enable_security_service_cert_authentication` variable is the key to enabling iSF certificate-based authentication. By setting this variable to `true`, you cause the server to perform iSF-level certificate authentication.
4. The IIOP/TLS target secure invocation policy must require `EstablishTrustInClient`. Evidently, if the client does not provide a certificate during the SSL/TLS handshake, there will be no certificate available to perform the iSF-level authentication.

Related administration tasks

When using X.509 certificate-based authentication, it is necessary to add the appropriate user data to your *enterprise security system* (which is integrated with the Orbix security service through an iSF adapter), as follows:

- File adapter (do not use in deployed systems)—see "Certificate-based authentication for the file adapter" on page 189
- LDAP adapter—see "Certificate-based authentication for the LDAP adapter" on page 190.

Caching of Credentials

Overview

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

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

Cache time-out

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

Cache size

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

Configuration variables

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

`plugins:gsp:authentication_cache_size`

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

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

`plugins:gsp:authentication_cache_timeout`

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

CHAPTER 5

Single Sign-On for CORBA Applications

Single sign-on (SSO) is an Orbix security feature which minimizes the exposure of usernames and passwords to snooping. After initially signing on, a client communicates with other applications by passing an SSO token in place of the original username and password.

In this chapter

This chapter discusses the following topics:

SSO and the Login Service

Overview

The SSO feature is implemented by the following elements of Orbix:

- *Login service*—a central service which can authenticate username/password combinations and generate SSO tokens.
- *GSP plug-in*—the generic security plug-in, which is embedded in a client application, is responsible for contacting the login service to obtain an SSO token.

Advantages of SSO

SSO greatly increases the security of an application in the Orbix Security Framework, offering the following advantages:

- Password visibility is restricted to the Login Service.
- Clients use SSO tokens to communicate with servers.
- Clients can be configured to use SSO with no code changes.
- SSO tokens are configured to expire after a specified length of time.
- When an SSO token expires, the CORBA client automatically requests a new token from the login service. No additional user code is required.

Embedded login service

Figure 20 shows an overview of the login service which, by default, is embedded in the same process as the Orbix security service. The client ORB automatically requests an SSO token by sending a username and a password to the login service. If the username and password are successfully authenticated, the login service returns an SSO token.

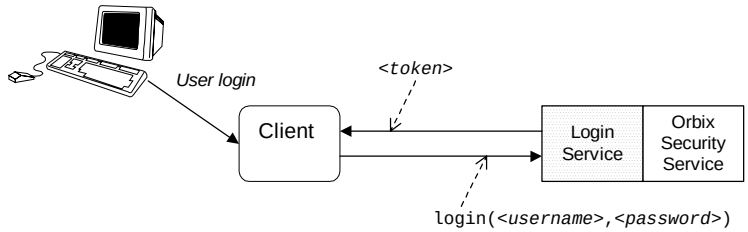

Figure 20: *Client Requesting an SSO Token from the Login Service*

SSO token

The SSO token is a compact key that the Orbix security service uses to access a user's session details, which are stored in a cache.

SSO token expiry

The Orbix security service is configured to impose the following kinds of timeout on an SSO token:

- *SSO session timeout*—this timeout places an absolute limit on the lifetime of an SSO token. When the timeout is exceeded, the token expires.
- *SSO session idle timeout*—this timeout places a limit on the amount of time that elapses between authentication requests involving the SSO token. If the central Orbix security service receives no authentication requests in this time, the token expires.

For more details, see "Configuring Single Sign-On Properties" on page 168.

Automatic token refresh

In theory, the expiry of SSO tokens could prove a nuisance to client applications, because servers will raise a `CORBA::NO_PERMISSION` exception whenever an SSO token expires. In practice, however, when SSO is enabled, the GSP plug-in catches the `NO_PERMISSION` exception on the client side and contacts the login service again to refresh the SSO token automatically. The GSP plug-in then automatically retries the failed operation invocation.

Connection to the login server

It is imperative that a connection to the login service is strongly protected by SSL/TLS, in order to avoid exposing usernames and passwords to snooping. Hence, by default, the client-to-login service connection is protected by strong SSL/TLS security policies and the IIOP/TLS client secure invocation policy requires the following association options:

```
["Integrity", "Confidentiality", "DetectReplay",
   "DetectMisordering", "EstablishTrustInTarget"];
```

This protection remains in force, irrespective of the association options set explicitly by the SSL/TLS client secure invocation policy.

Note: The only way to reduce the level of protection on login service connections is to set the `plugins:gsp:enforce_secure_comms_to_sso_server` variable to false.

Standalone login service

It is possible, in principle, to reconfigure the login service as a standalone server (that is, a standalone process that runs independently of the Orbix security service). Currently, however, the `itconfigure` utility can only generate domains with an embedded login service.

Please contact IONA Professional Services for more details:

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

Username/Password-Based SSO

Overview

This section describes how to configure a client so that it transmits an SSO token in place of a username and a password (that is, SSO is used in conjunction with the CSI authentication over transport mechanism).

CSI layers

The CSIv2 standard defines two layers for transmitting credentials:

- *CSI authentication over transport (GSSUP authentication)*—this layer is used to transmit username, password, and domain data which can then be authenticated on the server side.
- *CSI identity assertion*—this layer is used to transmit just a username (asserted identity). It is not needed for the scenarios in this section.

GSSUP authentication without SSO

Figure 21 gives an overview of Generic Security Service Username/Password (GSSUP) based authentication without SSO. In this case, the username, *<username>*, and password, *<password>*, are passed directly to the target server, which then contacts the Orbix security service to authenticate the username/password combination.

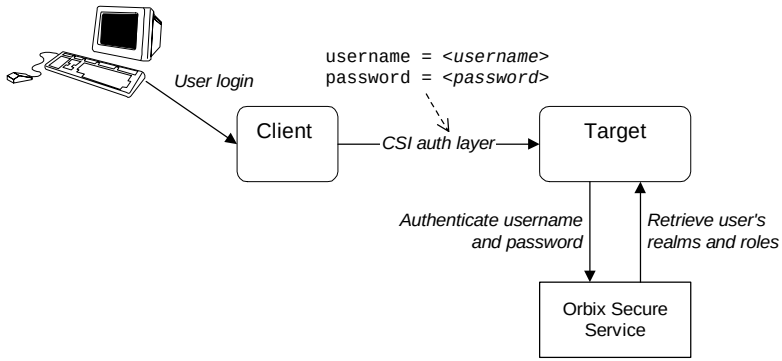

Figure 21: *Overview of GSSUP Authentication without SSO*

GSSUP authentication with SSO

Figure 22 gives an overview of username/password-based (GSSUP) authentication when SSO is enabled.

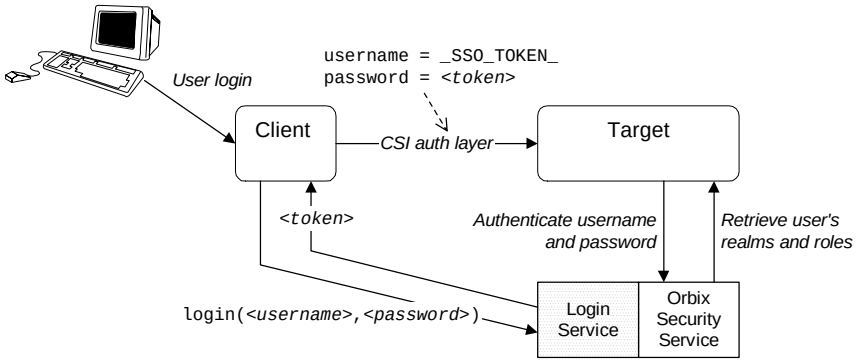

Figure 22: *Overview of GSSUP Authentication with SSO*

Prior to contacting the target server for the first time, the client ORB sends the username, *<username>*, and password, *<password>*, to the login server, getting an SSO token, *<token>* in return. The client ORB then includes a CSIv2 service context in the next request to the target server, sending the special string, `_SSO_TOKEN_`, instead of a username and the SSO token, *<token>*, instead of a password. The target server's ORB contacts the Orbix security service to authenticate the username/password combination and to obtain the user's authorization data.

Note: The target server is not aware whether the client has used the login service or not. It is the Orbix security service that knows to treat the `_SSO_TOKEN_` username in a special way.

Related configuration variables

The following variables are relevant to username/password-based SSO:

`plugins:gsp:enable_gssup_sso`
: Enables SSO with a username and a password (that is, GSSUP) when set to `true`.

`plugins:gsp: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. This policy is used to ensure that sensitive password information is seen only by a specific login server. For details on the syntax of certificate constraints, see "Applying Constraints to Certificates" on page 376.

Client configuration

Example 9 shows a typical configuration for an SSO client that employs GSSUP authentication.

Example 9: *Client Configuration for Username/Password-Based SSO*

```
# Orbix Configuration File
corba_login_server_test_with_tls
{
    orb_plugins = ["local_log_stream", "iiop_profile", "giop",
   "iiop_tls", "gsp"];

    event_log:filters = ["IT_GSP=*", "IT_CSI=*", "IT_TLS=*",
   "IT_IIOP_TLS=*", "IT_ATLI2_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", "GIOP+IIOP", "GIOP+IIOP_TLS"];

1   plugins:gsp:sso_server_certificate_constraints =
   ["C=US,ST=Massachusetts,O=ABigBank*,CN=Orbix2000 IONA
   Services*"];

    sso_client_gssup
    {
2       principal_sponsor:use_principal_sponsor = "false";
```

Example 9: *Client Configuration for Username/Password-Based SSO*

```
3       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", "EstablishTrustInTarget"];

4           plugins:csi:allow_csi_reply_without_service_context =
        "false";
5           policies:csi:auth_over_transport:client_supports =
        ["EstablishTrustInClient"];

6           principal_sponsor:csi:use_principal_sponsor = "true";
            principal_sponsor:csi:auth_method_id = "GSSUPMech";
            principal_sponsor:csi:auth_method_data =
        ["username=paulh", "password=password", "domain=PCGROUP"];

7           plugins:gsp:enable_gssup_sso = "true";
        };
};
```

The preceding client configuration can be described as follows:

1. The `plugins:gsp:sso_server_certificate_constraints` variable specifies certificate constraints that apply only to the X.509 certificate from the login server. If the login server's certificate fails to match these constraints, a `CORBA::NO_PERMISSION` exception is thrown on the client side.
2. In this example, the SSL/TLS principal sponsor is not used (the SSL/TLS principal sponsor is used to specify an application's own X.509 certificate credentials).

3. In this example, the client requires a secure SSL/TLS connection and requires the target server to authenticate itself with an X.509 certificate.

   Note: Irrespective of the level of security required by the these configuration settings, the SSO client *always* requires the login server connection to be secure and authenticated by an X.509 certificate. The only way you can reduce the level of security required by the login server connection is by setting the `plugins:gsp:enforce_secure_comms_to_sso_server` variable to `false`.

4. This setting enforces strict checking of reply messages from the server, to make sure the server actually supports CSIv2.
5. The CSI authentication over transport policy must support `EstablishTrustInClient` to enable the sending of usernames and passwords in CSIv2 service contexts.
6. The CSI principal sponsor, which specifies an application's own CSI credentials, can be enabled as shown here (alternatively, you could specify CSI credentials by programming; see "Creating CSIv2 Credentials" on page 470).

   In a deployed system, it is better to omit the password entry from the `principal_sponsor:csi:auth_method_data` setting. When omitted, the principal sponsor will prompt the user to enter a username and password as the client application starts up. The domain must be set to match the value of the `policies:csi:auth_over_transport:server_domain_name` variable on the server side.

   Note: Alternatively, you can specify the domain as an empty string, which would match any domain on the server side.

7. The `plugins:gsp:enable_gssup_sso` variable is set to `true` to enable the GSSUP single sign-on behavior.

Target configuration

Example 10 shows a typical configuration for a target server that accepts connections from clients that authenticate themselves using GSSUP.

Example 10: *Target Configuration for Username/Password-Based SSO*

```
# Orbix Configuration File
corba_login_server_test_with_tls
{
    orb_plugins = ["local_log_stream", "iiop_profile", "giop",
   "iiop_tls", "gsp"];

    event_log:filters = ["IT_GSP=*", "IT_CSI=*", "IT_TLS=*",
   "IT_IIOP_TLS=*", "IT_ATLI2_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", "GIOP+IIOP", "GIOP+IIOP_TLS"];

    plugins:gsp:sso_server_certificate_constraints =
   ["C=US,ST=Massachusetts,O=ABigBank*,CN=Orbix2000 IONA
   Services*"];

    server
    {
       policies:csi:auth_over_transport:authentication_service =
   "com.iona.corba.security.csi.AuthenticationService";

        principal_sponsor:use_principal_sponsor = "true";
        principal_sponsor:auth_method_id = "pkcs12_file";
1       principal_sponsor:auth_method_data =
   ["filename=W:\art\etc\tls\x509\certs\demos\bank_server.p12",
   "password=bankserverpass"];

        binding:server_binding_list = ["CSI+GSP", "CSI", "GSP"];

        initial_references:IS2Authorization:plugin =
   "it_is2_authorization";

        plugins:it_is2_authorization:ClassName =
   "com.iona.corba.security.authorization.IS2AuthorizationPlugIn
   ";
```

Example 10: *Target Configuration for Username/Password-Based SSO*

```
2        plugins:gsp:action_role_mapping_file =
    "file://W:\art\etc\tls\x509\..\..\..\..\art_svcs\etc\actionro
    lemapping_with_interfaces.xml";
         plugins:gsp:authorization_realm = "AuthzRealm";
         policies:csi:auth_over_transport:server_domain_name =
    "PCGROUP";

         auth_csi
         {
3
    policies:iiop_tls:target_secure_invocation_policy:supports =
    ["Integrity", "Confidentiality", "DetectReplay",
    "DetectMisordering", "EstablishTrustInTarget"];

    policies:iiop_tls:target_secure_invocation_policy:requires =
    ["Integrity", "Confidentiality", "DetectReplay",
    "DetectMisordering"];

4            policies:csi:auth_over_transport:target_requires =
    ["EstablishTrustInClient"];
             policies:csi:auth_over_transport:target_supports =
    ["EstablishTrustInClient"];
         };
     };
};
```

The preceding target configuration can be described as follows:

1. As usual for an SSL/TLS server, the SSL/TLS principal sponsor is used to specify the location of the server's own X.509 certificate.
2. The `action_role_mapping` configuration variable specifies the location of an action-role mapping that controls access to the IDL interfaces implemented by the server.
3. In this example, the server requires a secure SSL/TLS connection, but does not require the client to authenticate itself with an X.509 certificate.
4. It is essential for the target server to require and support the `EstablishTrustInClient` option for CSI authentication over transport. This ensures that the server receives a username and a password from the client in a CSIv2 service context.

Related administration tasks

For details of how to configure SSO token timeouts, see "Configuring Single Sign-On Properties" on page 168.

Three Tier Example with Identity Assertion

Overview

This section describes what happens when the two-tier username/password-based SSO example is extended by a third tier, which uses the CSI identity assertion mechanism.

This scenario has the following essential features:

- *Client to second tier*—the CSI authentication over transport mechanism (GSSUP authentication) is enabled and the client is configured to use single sign-on.
- *Second tier to third tier*—the CSI identity assertion mechanism is enabled between these tiers. SAML data (containing details of the client user's roles and realms) is propagated between these tiers.

Three-tier scenario with piggybacking

Figure 23 shows the outline of a single sign-on scenario where SAML role and realm data is piggybacked between the second and third tiers.

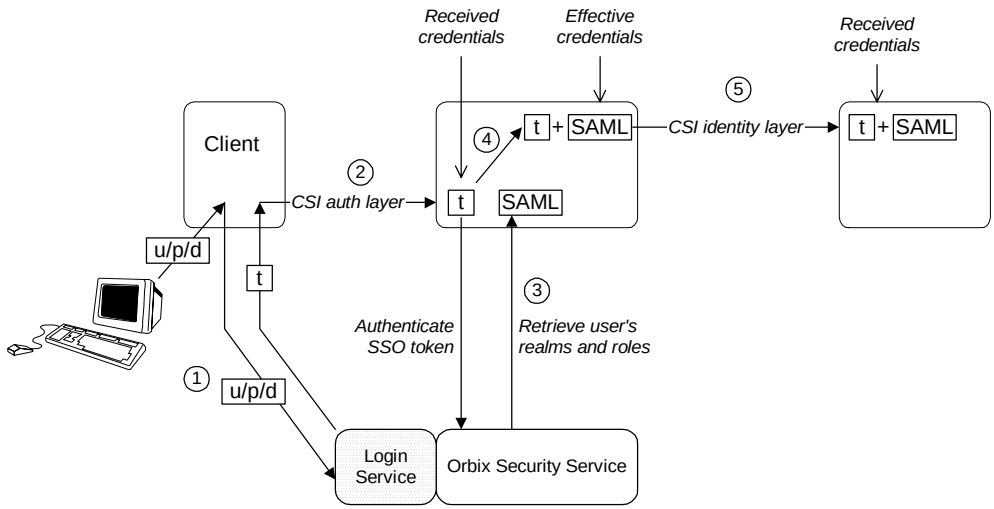

Figure 23: *Single Sign-On Scenario with Piggybacking Roles and Realms*

Steps

The operation invocations performed on behalf of the client shown in Figure 23 on page 111 can be described as follows:

Stage	Description
1	When single sign-on is enabled, the client calls out to the login service, passing in the client's GSSUP credentials, `u/p/d`, in order to obtain a single sign-on token, `t`.
2	When the client invokes an operation on the second-tier server, the SSO token, `t`, is sent as the password in the GSSUP authentication data. The GSSUP username has the reserved value `_SSO_TOKEN_`. The client SSO token, `t`, is now accessible through the `IT_CORBASEC::ExtendedReceivedCredentials` interface.
3	When the SSO token is received by the middle-tier server, it calls out to the Orbix security service to authenticate the client token and retrieve the SAML authorization data containing the user's complete role and realm data.
4	If the second tier now invokes an operation on the third tier, the *effective credentials* for the invocation are constructed as follows: • The client username is used as the asserted identity (to be propagated through the CSI identity assertion mechanism). • The client SSO token, `t`, from the received credentials is inserted into an IONA-proprietary service context.

Stage	Description
5	When the request message is sent to the third tier, the asserted identity is sent through the CSI identity layer, and the single sign-on token, `t`, is sent in an IONA-proprietary service context, accompanied by the SAML role and realm data. In the third tier, no call-out to the Orbix Security Service is required, because the SAML data includes all of the information needed for an authorization check. **WARNING:** It is *essential* that an adequate degree of trust is established between the third-tier server and the second-tier server. In this scenario, the third tier is completely dependent on the second tier to perform authentication on its behalf.

Configuration notes

The most important policy settings for this three-tier scenario with SAML piggybacking are briefly described here.

Client to Second Tier

The client is configured to support CSI authentication over transport and single sign-on with the following configuration settings (the `sso_server_certificate_constraints` setting would have to be customised to match your login server's X.509 certificate):

```
policies:csi:auth_over_transport:client_supports =
   ["EstablishTrustInClient];
plugins:gsp:enable_gssup_sso = "true";
plugins:gsp:sso_server_certificate_constraints =
   ["C=US,ST=Massachusetts,O=ABigBank*,CN=Orbix2000 IONA
   Services*"];
```

The second tier is configured to support CSI authentication over transport from incoming connections with the following settings:

```
policies:csi:auth_over_transport:target_supports =
   ["EstablishTrustInClient];
policies:csi:auth_over_transport:target_requires =
   ["EstablishTrustInClient];
```

Second Tier to Third Tier

The second tier is configured to support CSI identity assertion for outgoing connections with the following configuration settings:

```
policies:csi:attribute_service:client_supports =
   ["IdentityAssertion"];
```

The third tier is configured to support CSI identity assertion from incoming connections with the following settings:

```
policies:csi:attribute_service:target_supports =
   ["IdentityAssertion"];
```

X.509 Certificate-Based SSO

Overview

Normally, during certificate-based authentication, a client transmits its X.509 certificate during the SSL/TLS handshake. This certificate is then used for the authentication step with the Orbix security service (see "X.509 Certificate-Based Authentication" on page 91).

In contrast to this, in the SSO case a client transmits an SSO token through the CSI security layer (using CSI authentication over transport), having previously obtained the SSO token by authenticating its own certificate with the login server. The client's certificate might also be propagated directly to the target, in addition to the SSO token, but this would not be the usual case.

Certificate-based authentication without SSO

Figure 24 gives an overview of ordinary certificate-based authentication without SSO. In this case, the client's X.509 certificate is passed directly to the target server (during the SSL/TLS handshake). The target server then contacts the Orbix security service to authenticate the certificate.

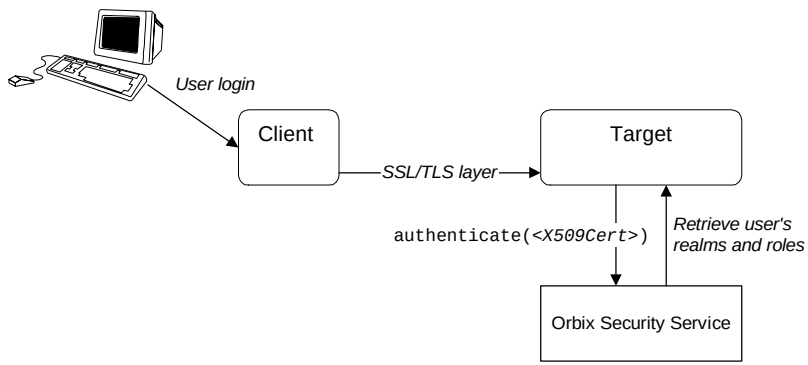

Figure 24: *Overview of Certificate-Based Authentication without SSO*

Certificate-based authentication with SSO

Figure 25 gives an overview of certificate-based authentication when SSO is enabled.

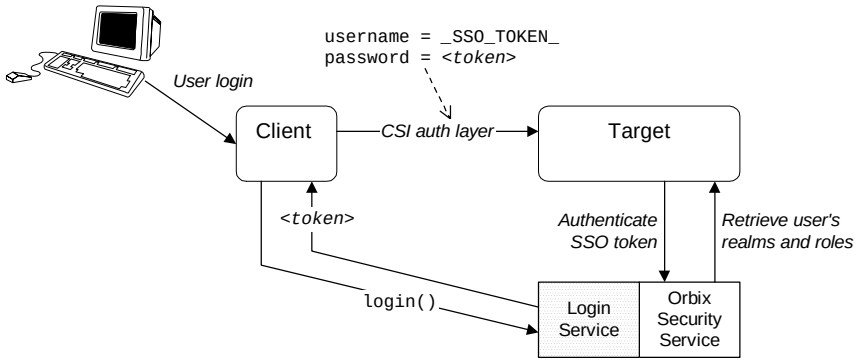

Figure 25: *Overview of Certificate-Based Authentication with SSO*

Prior to contacting the target server for the first time, the client ORB invokes the `login()` operation on the login server. The login server retrieves the client's X.509 certificate from the SSL/TLS received credentials, authenticates the certificate, and sends back an SSO token, *<token>* in return.

The client then sends a request to the target server, including the special username, `_SSO_TOKEN_`, and the password, *<token>*, in a CSIv2 service context. The target server contacts the Orbix security service to authenticate the username/password combination and to retrieve the user's authorization data (realms and roles).

Difference between username/password-based SSO and certificate-based SSO

The key difference between username/password-based SSO (Figure 22 on page 104) and certificate-based SSO (Figure 25) lies in the communication with the login server. In the username/password-based case, the client sends GSSUP data to be authenticated to the login service; whereas in the certificate-based case, the client sends an X.509 certificate to be authenticated to the login service.

There is no difference in the nature of the communication between the client and the target, however. In both cases, an SSO token is transmitted through the CSI authorization over transport layer.

Related configuration variables

The following variables are relevant to certificate-based SSO:

`plugins:gsp:enable_x509_sso`
: Enables certificate-based SSO when set to `true`.

`plugins:gsp: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 on the syntax of certificate constraints, see "Applying Constraints to Certificates" on page 376.

Typical scenario

The most likely scenario where you might need certificate-based SSO is where an existing server is configured to require username/password credentials, but you want to connect to the server using clients that have only X.509 certificate credentials. By enabling SSO on the client side, the clients acquire username/password credentials which the target server can then use for the purpose of authentication and authorization.

Client configuration

Example 11 shows a typical configuration for an SSO client that employs certificate-based authentication.

Example 11: *Client Configuration for Certificate-Based Authentication*

```
# Orbix Configuration File
corba_login_server_test_with_tls
{
    orb_plugins = ["local_log_stream", "iiop_profile", "giop",
   "iiop_tls", "gsp"];

    event_log:filters = ["IT_GSP=*", "IT_CSI=*", "IT_TLS=*",
   "IT_IIOP_TLS=*", "IT_ATLI2_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", "GIOP+IIOP", "GIOP+IIOP_TLS"];

1   plugins:gsp:sso_server_certificate_constraints =
   ["C=US,ST=Massachusetts,O=ABigBank*,CN=Orbix2000 IONA
   Services*"];
```

Example 11: *Client Configuration for Certificate-Based Authentication*

```
    sso_client_x509
    {
2
   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", "EstablishTrustInTarget"];

3       plugins:csi:allow_csi_reply_without_service_context =
   "false";
4       principal_sponsor:use_principal_sponsor = "true";
        principal_sponsor:auth_method_id = "pkcs12_file";
        principal_sponsor:auth_method_data =
   ["filename=W:\art\etc\tls\x509\certs\demos\bob.p12",
   "password=bobpass"];

5       policies:csi:auth_over_transport:client_supports =
   ["EstablishTrustInClient"];

6       plugins:gsp:enable_x509_sso = "true";
    };
};
```

The preceding client configuration can be described as follows:

1. The `plugins:gsp:sso_server_certificate_constraints` variable specifies certificate constraints that apply only to the X.509 certificate from the login server. If the login server's certificate fails to match these constraints, a `CORBA::NO_PERMISSION` exception is thrown on the client side.
2. In this example, the client requires a secure SSL/TLS connection and requires the target server to authenticate itself with an X.509 certificate. The client also supports the SSL/TLS `EstablishTrustInClient` option.

   Note: Irrespective of the level of security required by the these configuration settings, the SSO client *always* requires the login server connection to be secure and authenticated by an X.509 certificate. The only way you can reduce the level of security required by the login server connection is by setting the `plugins:gsp:enforce_secure_comms_to_sso_server` variable to `false`.

3. This setting enforces strict checking of reply messages from the server, to make sure the server actually supports CSIv2.
4. The client must have its own X.509 certificate to authenticate itself to the target. In this example, the SSL/TLS principal sponsor is used to specify the location of a PKCS#12 file containing the client's certificate.
5. The CSI authentication over transport policy must support `EstablishTrustInClient` to enable the sending of usernames and passwords in CSIv2 service contexts.
6. The `plugins:gsp:enable_x509_sso` variable is set to `true` to enable the X.509 single sign-on behavior.

Target configuration

Example 12 shows the configuration for a target server that requires GSSUP username/password credentials, but can also accept connections from clients that use X.509 certificate-based SSO.

Example 12: *Target Configuration for Certificate-Based Authentication*

```
# Orbix Configuration File
corba_login_server_test_with_tls
{
    orb_plugins = ["local_log_stream", "iiop_profile", "giop",
   "iiop_tls", "gsp"];

    event_log:filters = ["IT_GSP=*", "IT_CSI=*", "IT_TLS=*",
   "IT_IIOP_TLS=*", "IT_ATLI2_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", "GIOP+IIOP", "GIOP+IIOP_TLS"];

    plugins:gsp:sso_server_certificate_constraints =
   ["C=US,ST=Massachusetts,O=ABigBank*,CN=Orbix2000 IONA
   Services*"];

    server
    {
       policies:csi:auth_over_transport:authentication_service =
   "com.iona.corba.security.csi.AuthenticationService";

1       principal_sponsor:use_principal_sponsor = "true";
        principal_sponsor:auth_method_id = "pkcs12_file";
        principal_sponsor:auth_method_data =
   ["filename=W:\art\etc\tls\x509\certs\demos\bank_server.p12",
   "password=bankserverpass"];

        binding:server_binding_list = ["CSI+GSP", "CSI", "GSP"];

        initial_references:IS2Authorization:plugin =
   "it_is2_authorization";

        plugins:it_is2_authorization:ClassName =
   "com.iona.corba.security.authorization.IS2AuthorizationPlugIn
   ";
```

Example 12: *Target Configuration for Certificate-Based Authentication*

```
2        plugins:gsp:action_role_mapping_file =
    "file://W:\art\etc\tls\x509\..\..\..\..\art_svcs\etc\actionro
    lemapping_with_interfaces.xml";
         plugins:gsp:authorization_realm = "AuthzRealm";
         policies:csi:auth_over_transport:server_domain_name =
    "PCGROUP";

         require_gssup_support_x509_with_sso
         {

    policies:iiop_tls:target_secure_invocation_policy:supports =
    ["Integrity", "Confidentiality", "DetectReplay",
    "DetectMisordering", "EstablishTrustInTarget",
    "EstablishTrustInClient"];
3
    policies:iiop_tls:target_secure_invocation_policy:requires =
    ["Integrity", "Confidentiality", "DetectReplay",
    "DetectMisordering"];

4            policies:csi:auth_over_transport:target_requires =
    ["EstablishTrustInClient"];
             policies:csi:auth_over_transport:target_supports =
    ["EstablishTrustInClient"];
           };
     };
};
```

The preceding target configuration can be described as follows:

1. As usual for an SSL/TLS server, the SSL/TLS principal sponsor is used to specify the location of the server's own X.509 certificate.
2. The `action_role_mapping` configuration variable specifies the location of an action-role mapping that controls access to the IDL interfaces implemented by the server.
3. The server requires a secure SSL/TLS connection, but does not require the client to authenticate itself with an X.509 certificate.
4. Because the target server requires the `EstablishTrustInClient` option for CSI authentication over transport, clients must supply GSSUP username/password credentials. This condition is also satisfied by clients that use X.509 certificate-based SSO, because this results in the generation of GSSUP username/password credentials.

Related administration tasks

For details of how to configure SSO token timeouts, see "Configuring Single Sign-On Properties" on page 168.

Enabling Re-Authentication at Each Tier

Overview

This section describes a three-tier SSO scenario where piggybacking of SAML data (containing details of the client user's roles and realms) is disabled. This forces an SSO token to be re-authenticated at each tier in a multi-tier system, because the servers in each tier need to contact the Orbix security service to obtain the SAML data.

Advantages of enabling re-authentication

Re-enabling authentication at each tier has the following potential advantages:

- If your distributed application crosses different security domains, it might be necessary to re-authenticate credentials in a new domain.
- Sometimes, if the quantity of SAML data is very large, it might be more efficient for servers to retrieve the SAML data directly from the Orbix security service.

Disabling SAML piggybacking

There are two configuration variables that control SAML piggybacking.

`plugins:gsp:assert_authorization_info`

If `false`, SAML data is not sent on outgoing connections. Default is `true`.

`plugins:gsp:accept_asserted_authorization_info`

If `false`, SAML data is not read from incoming connections. Default is `true`.

Three-tier scenario without piggybacking

Figure 26 shows the outline of a single sign-on scenario where the propagation of SAML role and realm data is disabled.

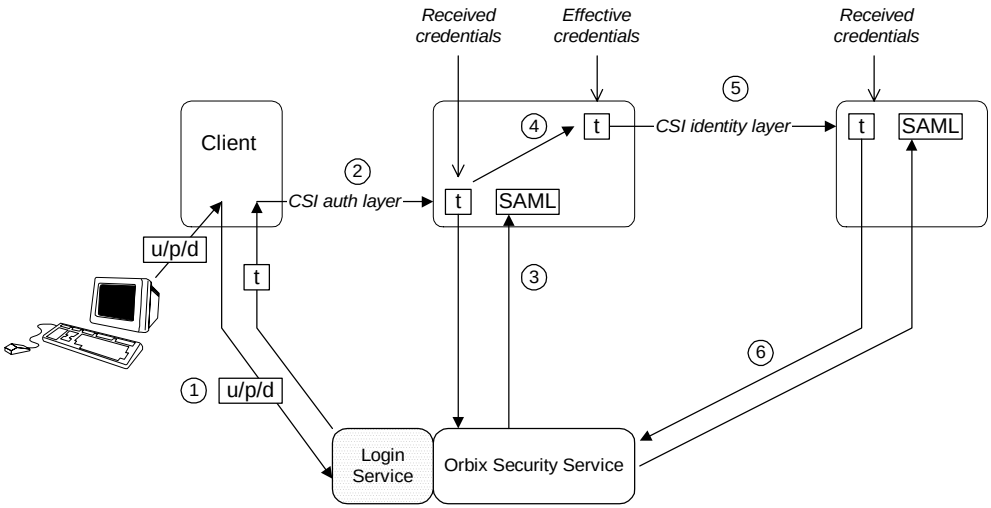

Figure 26: *Single Sign-On Scenario without Piggybacking Roles and Realms*

Steps

The operation invocations performed on behalf of the client shown in Figure 26 on page 124 can be described as follows:

Stage	Description
1	When single sign-on is enabled, the client calls out to the login service, passing in the client's GSSUP credentials, `u/p/d`, in order to obtain a single sign-on token, `t`.
2	When the client invokes an operation on the second-tier server, the SSO token, `t`, is sent as the password in the GSSUP username/password credentials.
3	The second tier re-authenticates the client's SSO token, `t`, by calling out to the Orbix Security Service. The return value contains the SAML role and realm data for the token.

Stage	Description
4	If the second tier now invokes an operation on the third tier, the *effective credentials* for the invocation are constructed as follows: • The client username is used as the asserted identity (to be propagated through the CSI identity assertion mechanism). • The client SSO token, `t`, from the received credentials is inserted into an IONA-proprietary service context.
5	When the request message is sent to the third tier, only the asserted identity and the single sign-on token, `t`, are included. Propagation of the SAML authorization data is disabled.
6	The third tier re-authenticates the client's SSO token, `t`, by calling out to the Orbix Security Service. The return value contains the SAML role and realm data for the token.

Configuration notes

The most important policy settings for this three-tier scenario without SAML piggybacking are briefly described here.

Client to Second Tier

The client is configured to support CSI authentication over transport and single sign-on without SAML piggybacking, with the following configuration settings (the `sso_server_certificate_constraints` setting would have to be customised to match your login server's X.509 certificate):

```
policies:csi:auth_over_transport:client_supports =
   ["EstablishTrustInClient];
plugins:gsp:enable_gssup_sso = "true";
plugins:gsp:sso_server_certificate_constraints =
   ["C=US,ST=Massachusetts,O=ABigBank*,CN=Orbix2000 IONA
   Services*"];
plugins:gsp:assert_authorization_info = "false";
```

The second tier is configured to support CSI authentication over transport from incoming connections, but not to accept SAML data, with the following settings:

```
policies:csi:auth_over_transport:target_supports =
   ["EstablishTrustInClient];
policies:csi:auth_over_transport:target_requires =
   ["EstablishTrustInClient];
plugins:gsp:accept_asserted_authorization_info = "false";
```

Second Tier to Third Tier

The second tier is configured to support CSI identity assertion for outgoing connections, but not to send SAML data, with the following configuration settings:

```
policies:csi:attribute_service:client_supports =
   ["IdentityAssertion"];
plugins:gsp:assert_authorization_info = "false";
```

The third tier is configured to support CSI identity assertion from incoming connections, but not to accept SAML data, with the following settings:

```
policies:csi:attribute_service:target_supports =
   ["IdentityAssertion"];
plugins:gsp:accept_asserted_authorization_info = "false";
```

Optimising Retrieval of Realm Data

Overview

By default, when the GSP plug-in connects to the security service to authenticate a user's security credentials, it retrieves all of the realm and role data for that user. For example, if a user has security data for realms, A, B, and C, the authentication step would return realm and role data for each of the three realms, A, B, and C.

In an enterprise system, the amount of realm data assocated with each user might become very large. In such systems, it is desirable to optimize the authentication step by returning only the realm data that is needed at a particular point in the system, rather than retrieving all of the realm data at once. Orbix enables you to restrict the amount of realm data returned at the authentication step by enabling a feature known as *realm filtering*.

Enabling realm filtering

To enable realm filtering, set the following configuration variable to `false`:

```
plugins:gsp:retrieve_isf_auth_principal_info_for_all_realms
```

By default, the GSP plug-in would retrieve a user's role and realm data for *all* realms when contacting the security service. When realm filtering is enabled in an Orbix server, however, the GSP plug-in checks to see whether the following configuration variable is set:

```
plugins:gsp:authorization_realm
```

If the preceding variable is set to a specific realm, the GSP plug-in proceeds to retrieve realm and role data for that realm only.

Same-realm scenario

Figure 27 shows an example of realm filtering applied to a three-tier system, where the intermediate server and the target server both belong to the *same* realm, A. In this case, the realm filtering optimization works effectively, because the target server can re-use the role and realm data (SAML-A data) obtained by the intermediate server.

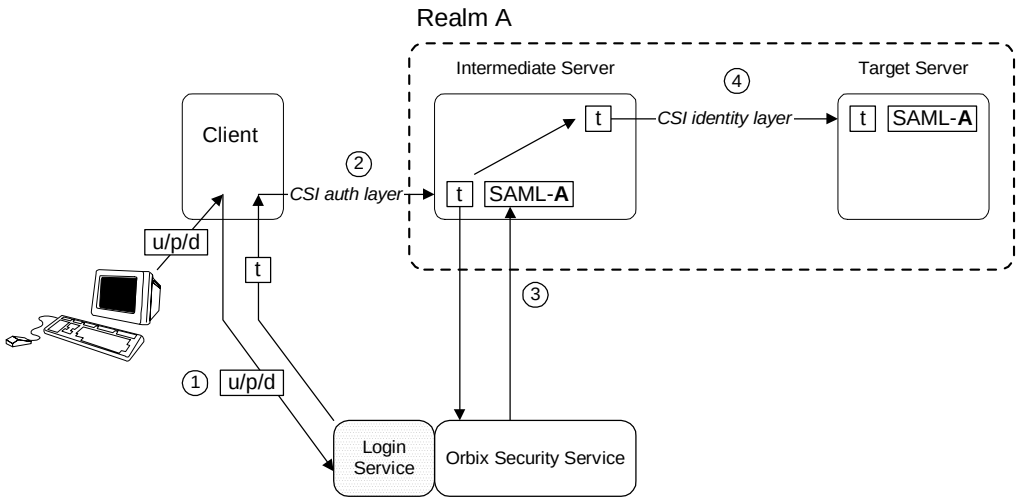

Figure 27: *Intermediate and Target Belong to Same Realm*

Same-realm stages

The same-realm scenario shown in Figure 27 can be described as follows:

Stage	Description
1	The client calls out to the login service, passing in the client's GSSUP credentials, `u/p/d`, in order to obtain a single sign-on token, `t`.
2	When the client invokes an operation on the intermediate server, the SSO token, `t`, is included with the request message (in the CSI authentication layer).

Stage	Description
3	The intermediate server re-authenticates the client's SSO token, `t`, by calling out to the Orbix Security Service. Because the intermediate server is configured to use realm filtering, it requests SAML role and realm data for realm A *only*.
4	The intermediate server invokes an operation on the target server. The request message includes the client SSO token, `t`, and the SAML data for realm A, `SAML-A`. Because the target server also belongs to realm A, it can use the SAML data received from the intermediate server to make an access decision. It does not need to re-authenticate the token.

Same-realm configuration

Example 13 shows an outline of the configuration required for the same-realm scenario. The intermediate server is configured to use realm filtering by setting the `plugins:gsp:retrieve_isf_auth_principal_info_for_all_realms` variable to `false`. Both the intermediate and the target are configured to belong to realm A.

Example 13: *Same-Realm Scenario Configuration*

```
# Orbix Configuration File
client {
    ...
};

intermediate_server {
    ...
    plugins:gsp:retrieve_isf_auth_principal_info_for_all_realms
   = "false";
    plugins:gsp:authorization_realm = "A";
};

target_server {
    ...
    plugins:gsp:authorization_realm = "A";
};
```

Different-realm scenario

Figure 28 shows an example of realm filtering applied to a three-tier system, where the intermediate server and the target server belong to *different* realms, A and B. In this case, realm filtering does not provide an optimization and the target server must be configured to re-authenticate any incoming tokens.

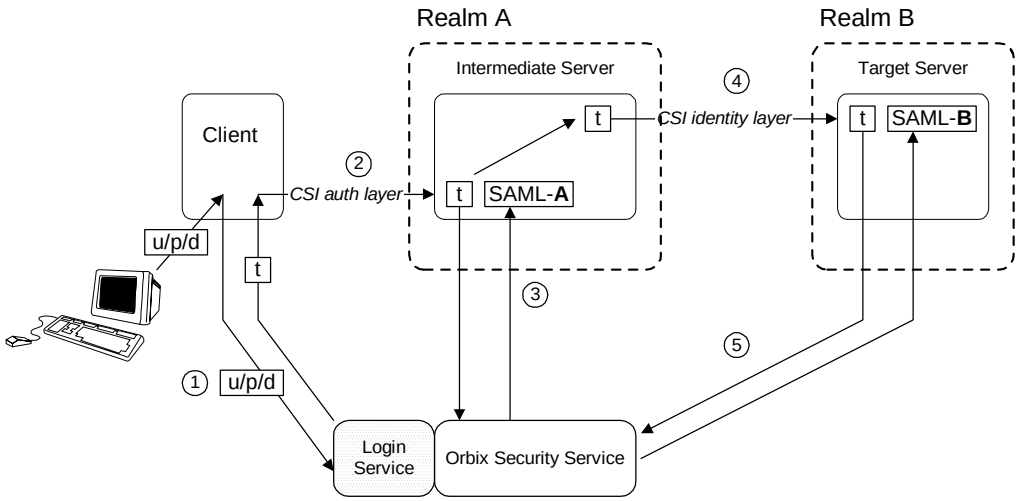

Figure 28: *Intermediate and Target Belong to Different Realms*

Different-realm stages

The different-realm scenario shown in Figure 28 can be described as follows:

Stage	Description
1	The client calls out to the login service, passing in the client's GSSUP credentials, `u/p/d`, in order to obtain a single sign-on token, `t`.
2	When the client invokes an operation on the intermediate server, the SSO token, `t`, is included with the request message (in the CSI authentication layer).

Stage	Description
3	The intermediate server re-authenticates the client's SSO token, `t`, by calling out to the Orbix Security Service. Because the intermediate server is configured to use realm filtering, it requests SAML role and realm data for realm A *only*.
4	The intermediate server invokes an operation on the target server. The request message includes the client SSO token, `t`, and the SAML data for realm A, `SAML-A`. The SAML data for realm A is of no use to the target server, which belongs to realm B. Therefore, the target server is configured to reject the transmitted realm data (that is, `plugins:gsp:accept_asserted_authorization_info` is set to `false`).
5	The target server re-authenticates the client's SSO token, `t`, to obtain the SAML role and realm data for realm B.

Different-realm configuration

Example 14 shows an outline of the configuration required for the different-realm scenario. Both the intermediate server and the target server are configured to use realm filtering by setting the `plugins:gsp:retrieve_isf_auth_principal_info_for_all_realms` variable to `false`. The intermediate and the target belong, however, to different realms: while the intermediate belongs to realm A, the target belongs to realm B. To force the target server to re-authenticate incoming tokens (and thus retrieve the necessary SAML data for realm B), the target server configuration sets `plugins:gsp:accept_asserted_authorization_info` to `false`.

Example 14: *Different-Realm Scenario Configuration*

```
# Orbix Configuration File
client {
    ...
};

intermediate_server {
    ...
    plugins:gsp:retrieve_isf_auth_principal_info_for_all_realms
   = "false";
```

Example 14: *Different-Realm Scenario Configuration*

```
    plugins:gsp:authorization_realm = "A";
};

target_server {
    ...
    plugins:gsp:retrieve_isf_auth_principal_info_for_all_realms
   = "false";
    plugins:gsp:authorization_realm = "B";
    plugins:gsp:accept_asserted_authorization_info = "false";
};
```

SSO Sample Configurations

Overview

This section provides SSO sample configurations that show how to configure the client side and the server side in a variety of different ways.

Client SSO configurations

The following client configurations appear in Example 15:

- `sso_client_x509`—configuration for an SSO client that uses X.509 certificate-based SSO credentials to authenticate itself to the server.
- `sso_client_gssup`—configuration for an SSO client that provides username and password (GSSUP)-based SSO credentials to authenticate itself to the server.
- `sso_client_gssup_x509`—configuration for an SSO client that can authenticate itself to a server using *either* username/password-based SSO credentials *or* X.509 certificate-based SSO credentials, depending on the requirements of the server.

Server SSO configurations

The following server configurations appear in Example 15:

- `auth_csi`—configuration for a server that requires the client to provide credentials over CSI. Three client scenarios are supported by this server configuration, as follows:
  - Client with username/password credentials (SSO not enabled).
  - Client with username/password-based SSO credentials.
  - Client with X.509 certificate-based SSO credentials.
- `auth_csi_and_x509`—configuration for a server that requires both X.509 certificate credentials (over SSL/TLS) and username/password credentials (over CSIv2). The following client scenarios are supported by this server configuration:
  - Client with both X.509 certificate credentials and username/password credentials (SSO not enabled).
  - Client with X.509 certificate-based SSO credentials.
  - Client with *both* X.509 certificate credentials *and* username/password-based SSO credentials.

- Client with *both* X.509 certificate-based SSO credentials *and* username/password-based SSO credentials (for example, the `sso_client_gssup_x509` configuration scope). In this case, the client would store three different kinds of credentials: X.509 certificate credentials, X.509 certificate-based SSO credentials, and username/password-based SSO credentials. Only two of the stored credentials would actually be used when communicating with the server (X.509 certificate credentials over SSL/TLS, and one of the SSO credentials over CSIv2).

SSO configuration examples

Example 15 shows a series of sample configurations suitable for SSO clients and SSO servers, supporting either GSSUP authentication, or X.509 certificate authentication, or both.

Example 15: *SSO Client and Server Configuration Examples*

```
# Orbix Configuration File
corba_login_server_test_with_tls
{
    principal_sponsor:use_principal_sponsor = "false";

    orb_plugins = ["local_log_stream", "iiop_profile", "giop",
   "iiop_tls", "gsp"];

    event_log:filters = ["IT_GSP=*", "IT_CSI=*", "IT_TLS=*",
   "IT_IIOP_TLS=*", "IT_ATLI2_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", "GIOP+IIOP", "GIOP+IIOP_TLS"];

    plugins:gsp:sso_server_certificate_constraints =
   ["C=US,ST=Massachusetts,O=ABigBank*,CN=Orbix2000 IONA
   Services*"];

    sso_client_x509
    {
```

Example 15: *SSO Client and Server Configuration Examples*

```
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=W:\art\etc\tls\x509\certs\demos\bob.p12",
"password=bobpass"];

     plugins:csi:allow_csi_reply_without_service_context =
"false";
     policies:csi:auth_over_transport:client_supports =
["EstablishTrustInClient"];

     plugins:gsp:enable_x509_sso = "true";
 };

 sso_client_gssup
 {

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"];

     plugins:csi:allow_csi_reply_without_service_context =
"false";
     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=paulh", "password=password", "domain=PCGROUP"];
```

Example 15: *SSO Client and Server Configuration Examples*

```
        plugins:gsp:enable_gssup_sso = "true";
    };

    sso_client_gssup_x509
    {

   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=W:\art\etc\tls\x509\certs\demos\bob.p12",
   "password=bobpass"];

        plugins:csi:allow_csi_reply_without_service_context =
   "false";
        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=paulh", "password=password", "domain=PCGROUP"];

        plugins:gsp:enable_gssup_sso = "true";
        plugins:gsp:enable_x509_sso = "true";
    };

    server
    {
       policies:csi:auth_over_transport:authentication_service =
   "com.iona.corba.security.csi.AuthenticationService";

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

Example 15: *SSO Client and Server Configuration Examples*

```
     binding:server_binding_list = ["CSI+GSP", "CSI", "GSP"];

     initial_references:IS2Authorization:plugin =
"it_is2_authorization";

     plugins:it_is2_authorization:ClassName =
"com.iona.corba.security.authorization.IS2AuthorizationPlugIn
";

     plugins:gsp:action_role_mapping_file =
"file://W:\art\etc\tls\x509\..\..\..\..\art_svcs\etc\actionro
lemapping_with_interfaces.xml";
     plugins:gsp:authorization_realm = "AuthzRealm";
     policies:csi:auth_over_transport:server_domain_name =
"PCGROUP";

     auth_csi
     {

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

policies:iiop_tls:target_secure_invocation_policy:requires =
["Integrity", "Confidentiality", "DetectReplay",
"DetectMisordering"];

         policies:csi:auth_over_transport:target_requires =
["EstablishTrustInClient"];
         policies:csi:auth_over_transport:target_supports =
["EstablishTrustInClient"];
     };

     auth_csi_and_x509
     {


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

Example 15: *SSO Client and Server Configuration Examples*

```
   policies:iiop_tls:target_secure_invocation_policy:requires =
   ["Integrity", "Confidentiality", "DetectReplay",
   "DetectMisordering", "EstablishTrustInClient"];

            policies:csi:auth_over_transport:target_requires =
   ["EstablishTrustInClient"];
            policies:csi:auth_over_transport:target_supports =
   ["EstablishTrustInClient"];
        };
    };
};
```

Part II

Orbix Security Framework Administration

In this part

This part contains the following chapters:

CHAPTER 6

Configuring the Orbix Security Service

This chapter describes how to configure the properties of the Orbix security service and, in particular, how to configure a variety of adapters that can integrate the Orbix security service with third-party enterprise security back-ends (for example, LDAP).

In this chapter

This chapter discusses the following topics:

Configuring the File Adapter

Overview

The iSF file adapter enables you to store information about users, roles, and realms in a flat file, a *security information file*. The file adapter is easy to set up and configure, but is appropriate for demonstration purposes only. This section describes how to set up and configure the iSF file adapter.

WARNING: The file adapter is provided for demonstration purposes only. IONA does not support the use of the file adapter in a production environment.

File locations

The following files configure the iSF file adapter:

- `is2.properties` file—the default location of the iSF properties file is as follows:
  ASPInstallDir`/etc/domains/`*DomainName*`/is2.properties`
  See "iS2 Properties File" on page 515 for details of how to customize the default iSF properties file location.
- Security information file—this file's location is specified by the `com.iona.isp.adapter.file.param.filename` property in the `is2.properties` file.

File adapter properties

Example 16 shows the properties to set for a file adapter.

Example 16: *Sample File Adapter Properties*

```
1  com.iona.isp.adapters=file

   ##########################################
   ##
   ## Demo File Adapter Properties
   ##
   ##########################################
2  com.iona.isp.adapter.file.class=com.iona.security.is2adapter.fil
      e.FileAuthAdapter
3  com.iona.isp.adapter.file.param.filename=ASPInstallDir/etc/domain
      s/DomainName/is2_user_password_role_file.txt
```

Example 16: *Sample File Adapter Properties*

```
##########################################
## General Orbix Security Service Properties
##########################################
# ... Generic properties not shown here ...
```

The necessary properties for a file adapter are described as follows:

1. Set `com.iona.isp.adapters=file` to instruct the Orbix security service to load the file adapter.
2. The `com.iona.isp.adapter.file.class` property specifies the class that implements the iSF file adapter.
3. The `com.iona.isp.adapter.file.param.filename` property specifies the location of the security information file, which contains information about users and roles.

   See "Managing a File Security Domain" on page 187 for details of how to create or modify the security information file.
4. *(Optionally)* You might also want to edit the general Orbix security service properties.

   See "Additional Security Configuration" on page 167 for details.

Configuring the LDAP Adapter

Overview

The IONA security platform integrates with the Lightweight Directory Access Protocol (LDAP) enterprise security infrastructure by using an LDAP adapter. The LDAP adapter is configured in an `is2.properties` file. This section discusses the following topics:

- Prerequisites
- File location.
- Minimal LDAP configuration.
- Basic LDAP properties.
- LDAP.param properties.
- LDAP server replicas.
- Logging on to an LDAP server.

Prerequisites

Before configuring the LDAP adapter, you must have an LDAP security system installed and running on your system. LDAP is *not* a standard part of Orbix E2A Application Server Platform, but you can use the Orbix security service's LDAP adapter with any LDAP v.3 compatible system.

File location

The following file configures the LDAP adapter:

- `is2.properties` file—the default location of the iSF properties file is as follows:
  ASPInstallDir`/etc/domains/`*DomainName*`/is2.properties`
  See "iS2 Properties File" on page 515 for details of how to customize the default iSF properties file location.

Minimal LDAP configuration

Example 17 shows the minimum set of iSF properties that can be used to configure an LDAP adapter.

Example 17: *A Sample LDAP Adapter Configuration File*

```
1  com.iona.isp.adapters=LDAP
   ##############################################
   ##
   ## LDAP Adapter Properties
   ##
   ##############################################
2  com.iona.isp.adapter.LDAP.class=com.iona.security.is2adapter.lda
      p.LdapAdapter


3  com.iona.isp.adapter.LDAP.param.host.1=10.81.1.400
   com.iona.isp.adapter.LDAP.param.port.1=389

4  com.iona.isp.adapter.LDAP.param.UserNameAttr=uid
   com.iona.isp.adapter.LDAP.param.UserBaseDN=dc=iona,dc=com
   com.iona.isp.adapter.LDAP.param.UserObjectClass=organizationalPe
      rson
   com.iona.isp.adapter.LDAP.param.UserSearchScope=SUB

5  com.iona.isp.adapter.LDAP.param.UserRoleDNAttr=nsroledn
   com.iona.isp.adapter.LDAP.param.RoleNameAttr=cn

6  com.iona.isp.adapter.LDAP.param.GroupNameAttr=cn
   com.iona.isp.adapter.LDAP.param.GroupObjectClass=groupofuniquena
      mes
   com.iona.isp.adapter.LDAP.param.GroupSearchScope=SUB
   com.iona.isp.adapter.LDAP.param.GroupBaseDN=dc=iona,dc=com
   com.iona.isp.adapter.LDAP.param.MemberDNAttr=uniqueMember

7  com.iona.isp.adapter.LDAP.param.version=3
```

The necessary properties for an LDAP adapter are described as follows:

1. Set `com.iona.isp.adapters=LDAP` to instruct the IONA Security Platform to load the LDAP adapter.
2. The `com.iona.isp.adapter.file.class` property specifies the class that implements the LDAP adapter.

3. For each LDAP server replica, you must specify the host and port where the LDAP server can be contacted. In this example, the host and port parameters for the primary LDAP server, `host.1` and `port.1`, are specified.
4. These properties specify how the LDAP adapter finds a user name within the LDAP directory schema. The properties are interpreted as follows:

`UserNameAttr`	The attribute type whose corresponding value uniquely identifies the user.
`UserBaseDN`	The base DN of the tree in the LDAP directory that stores user object class instances.
`UserObjectClass`	The attribute type for the object class that stores users.
`UserSearchScope`	The user search scope specifies the search depth relative to the user base DN in the LDAP directory tree. Possible values are: `BASE`, `ONE`, or `SUB`.

   See "iS2 Properties File" on page 515 for more details.
5. The following properties specify how the adapter extracts a user's role from the LDAP directory schema:

`UserRoleDNAttr`	The attribute type that stores a user's role DN.
`RoleNameAttr`	The attribute type that the LDAP server uses to store the role name.

6. These properties specify how the LDAP adapter finds a group name within the LDAP directory schema. The properties are interpreted as follows:

`GroupNameAttr`	The attribute type whose corresponding attribute value gives the name of the user group.
`GroupBaseDN`	The base DN of the tree in the LDAP directory that stores user groups.
`GroupObjectClass`	The object class that applies to user group entries in the LDAP directory structure.

`GroupSearchScope`	The group search scope specifies the search depth relative to the group base DN in the LDAP directory tree. Possible values are: `BASE`, `ONE`, or `SUB`.
`MemberDNAttr`	The attribute type that is used to retrieve LDAP group members.

See "iS2 Properties File" on page 515 for more details.

7. The LDAP version number can be either `2` or `3`, corresponding to LDAP v.2 or LDAP v.3 respectively.

Basic LDAP properties

The following properties must always be set as part of the LDAP adapter configuration:

```
com.iona.isp.adapters=LDAP
com.iona.isp.adapter.LDAP.class=com.iona.security.is2adapter.ldap
    .LdapAdapter
```

In addition to these basic properties, you must also set a number of LDAP parameters, which are prefixed by `com.iona.isp.adapter.LDAP.param`.

LDAP.param properties

Table 3 shows all of the LDAP adapter properties from the `com.iona.isp.adapter.LDAP.param` scope. Required properties are shown in bold:

Table 3: *LDAP Properties in the com.iona.isp.adapter.LDAP.param Scope*

LDAP Server Properties	LDAP User/Role Configuration Properties

`host.`<*Index*>
`port.`<*Index*>
`SSLEnabled.`<*Index*>
`SSLCACertDir.`<*Index*>
`SSLClientCertFile.`<*Index*>
`SSLClientCertPassword.`<*Index*>
`PrincipalUserDN.`<*Index*>
`PrincipalUserPassword.`<*Index*>
	`UserNameAttr`
	`UserBaseDN`
	`UserObjectClass`
	`UserSearchScope`
	`UserSearchFilter`
	`UserRoleDNAttr`
	`RoleNameAttr`
	`UserCertAttrName`

LDAP Group/Member Configuration Properties	**Other LDAP Properties**

`GroupNameAttr`
`GroupObjectClass`
`GroupSearchScope`
`GroupBaseDN`
`MemberDNAttr`
`MemberFilter`
	`MaxConnectionPoolSize`
	`version`
	`UseGroupAsRole`
	`RetrieveAuthInfo`
	`CacheSize`
	`CacheTimeToLive`

LDAP server replicas

The LDAP adapter is capable of failing over to one or more backup replicas of the LDAP server. Hence, properties such as `host.`<*Index*> and `port.`<*Index*> include a replica index as part of the parameter name.

For example, `host.1` and `port.1` refer to the host and port of the primary LDAP server, while `host.2` and `port.2` would refer to the host and port of an LDAP backup server.

Logging on to an LDAP server

The following properties can be used to configure login parameters for the <*Index*> LDAP server replica:

`PrincipalUserDN.`<*Index*>
`PrincipalUserPassword.`<*Index*>

The properties need only be set if the LDAP server is configured to require username/password authentication.

Secure connection to an LDAP server

The following properties can be used to configure SSL/TLS security for the connection between the Orbix security service and the <*Index*> LDAP server replica:

`SSLEnabled.`<*Index*>
`SSLCACertDir.`<*Index*>
`SSLClientCertFile.`<*Index*>
`SSLClientCertPassword.`<*Index*>

The properties need only be set if the LDAP server requires SSL/TLS mutual authentication.

iSF properties reference

For more details about the Orbix security service properties, see "iS2 Configuration" on page 513.

Clustering and Federation

Overview

Clustering and federation are two distinct, but related, features of the Orbix security service. Briefly, these features can be described as follows:

- *Clustering*—involves running several instances of the Orbix security service to provide what is effectively a single service. By running multiple security service instances as a *cluster*, Orbix enables you to support fault tolerance and replication features. Typically, in this case all of the security services in a cluster are integrated with a single authentication database back-end.
- *Federation*—enables SSO tokens to be recognized across multiple security domains. Each security domain is served by a distinct security service instance and each security service is integrated with a different database back-end.

In this section

This section contains the following subsections:

Federating the Orbix Security Service

Overview

Federation is meant to be used in deployment scenarios where there is more than one instance of an Orbix security service. By configuring the Orbix security service instances as a federation, the security services can talk to each other and access each other's session caches. Federation frequently becomes necessary when single sign-on (SSO) is used, because an SSO token can be verified only by the security service instance that originally generated it.

Federation is not clustering

Federation is not the same thing as clustering. In a federated system, user data is not replicated across different security service instances and there are no fault tolerance features provided.

Example federation scenario

Consider a simple federation scenario consisting of two security domains, each with their own Orbix security service instances, as follows:

- *First LDAP security domain*—consists of an Orbix security service (with `is2.current.server.id` property equal to `1`) configured to store user data in an LDAP database. The domain includes any Orbix applications that use this Orbix security service (ID=1) to verify credentials.

  In this domain, a login server is deployed which enables clients to use single sign-on.
- *Second LDAP security domain*—consists of an Orbix security service (with `is2.current.server.id` property equal to `2`) configured to store user data in an LDAP database. The domain includes any Orbix applications that use this Orbix security service (ID=2) to verify credentials.

The two Orbix security service instances are federated, using the configuration described later in this section. With federation enabled, it is possible for single sign-on clients to make invocations that cross security domain boundaries.

Federation scenario

Figure 29 shows a typical scenario that illustrates how iSF federation might be used in the context of an Orbix system.

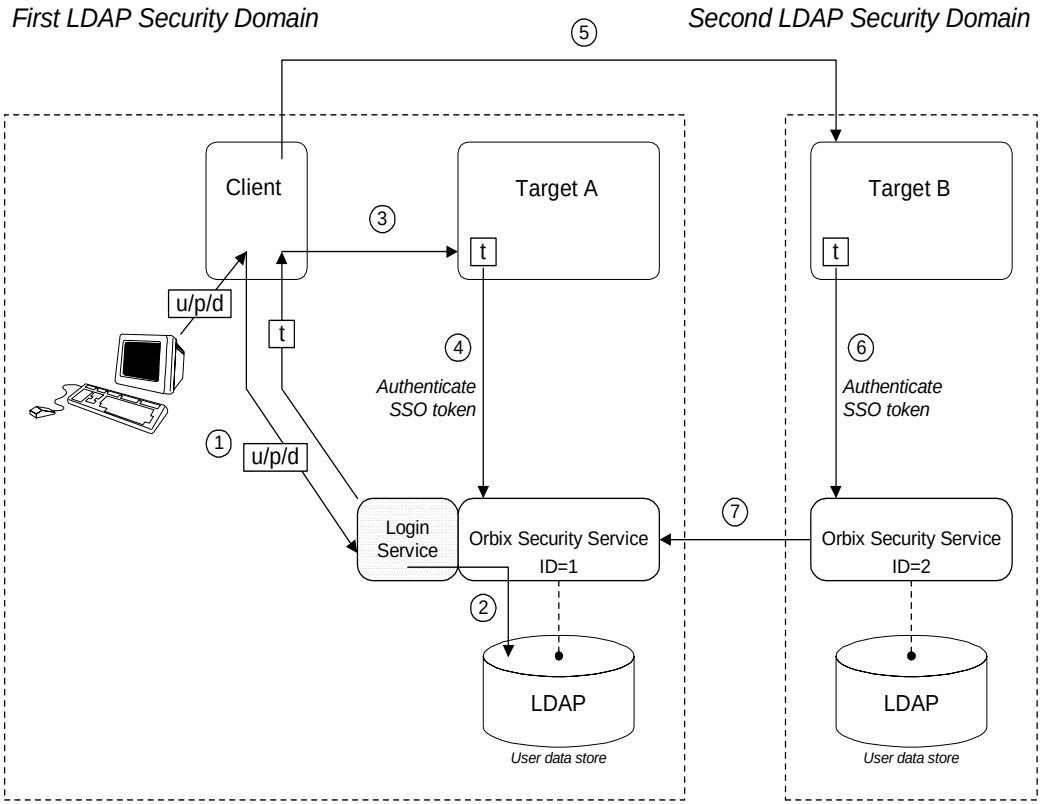

Figure 29: *An iSF Federation Scenario*

Federation scenario steps

The federation scenario in Figure 29 can be described as follows:

Stage	Description
1	With single sign-on (SSO) enabled, the client calls out to the login service, passing in the client's GSSUP credentials, `u/p/d`, in order to obtain an SSO token, `t`.
2	The login service delegates authentication to the Orbix security server (ID=1), which retrieves the user's account data from the LDAP backend.
3	The client invokes an operation on the *Target A*, belonging to the first LDAP security domain. The SSO token, `t`, is included in the message.
4	*Target A* passes the SSO token to the Orbix security server (ID=1) to be authenticated. If authentication is successful, the operation is allowed to proceed.
5	Subsequently, the client invokes an operation on the *Target B*, belonging to the second LDAP security domain. The SSO token, `t`, obtained in step **1** is included in the message.
6	*Target B* passes the SSO token to the second Orbix security server (ID=2) to be authenticated.
7	The second Orbix security server examines the SSO token. Because the SSO token is tagged with the first Orbix security server's ID (ID=1), verification of the token is delegated to the first Orbix security server. The second Orbix security server opens an IIOP/TLS connection to the first Orbix security service to verify the token.

Configuring the is2.properties files

Each instance of the Orbix security service should have its own `is2.properties` file. Within each `is2.properties` file, you should set the following:

- `is2.current.server.id`—a unique ID for this Orbix security service instance,
- `is2.cluster.properties.filename`—a shared cluster file.
- `is2.sso.remote.token.cached`—a boolean property enables caching of remote token credentials in a federated system.

  With caching enabled, the call from one federated security service to another (step 7 of Figure 29 on page 152) is only necessary to authenticate a token for the first time. For subsequent authentications, the security service (with ID=2) can obtain the token's security data from its own token cache.

For example, the first Orbix security server instance from Figure 29 on page 152 could be configured as follows:

```
# iS2 Properties File, for Server ID=1
...
##########################################
## iSF federation related properties
##########################################
is2.current.server.id=1
is2.cluster.properties.filename=C:/is2_config/cluster.properties
is2.sso.remote.token.cached=true
...
```

And the second Orbix security server instance from Figure 29 on page 152 could be configured as follows:

```
# iS2 Properties File, for Server ID=2
...
##########################################
## iSF federation related properties
##########################################
is2.current.server.id=2
is2.cluster.properties.filename=C:/is2_config/cluster.properties
is2.sso.remote.token.cached=true
...
```

Configuring the cluster properties file

All the Orbix security server instances within a federation should share a cluster properties file. For example, the following extract from the `cluster.properties` file shows how to configure the pair of embedded Orbix security servers shown in Figure 29 on page 152.

```
# Advertise the locations of the security services in the cluster.
com.iona.security.common.securityInstanceURL.1=corbaloc:it_iiops:1.2@security_ldap1:5001/IT_Secu
   rityService
com.iona.security.common.securityInstanceURL.2=corbaloc:it_iiops:1.2@security_ldap2:5002/IT_Secu
   rityService
```

This assumes that the first security service (ID=1) runs on host `security_ldap1` and IP port 5001; the second security service (ID=2) runs on host `security_ldap2` and IP port 5002. To discover the appropriate host and port settings for the security services, check the `plugins:security:iiop_tls` settings in the relevant configuration scope in the relevant Orbix configuration file for each federated security service.

The `securityInstanceURL.`*ServerID* variable advertises the location of a security service in the cluster. Normally, the most convenient way to set these values is to use the `corbaloc` URL format.

Sample cluster properties file

If you have generated a secure configuration domain, *DomainName*, on a host, *HostName*, you can then find a sample `cluster.properties` file in the following directory:

OrbixInstallDir`/etc/domains/`*DomainName*`/security_`*HostName*`/`

Failover and Replication

Overview

To support *high availability* of the Orbix security service, Orbix implements the following features:

- *Failover*—the security service is contacted using an IOR that contains the address of *every* security service in a cluster. Hence, if one of the services in the cluster crashes, or otherwise becomes unavailable, an application can automatically try one of the alternative addresses listed in the IOR.
- *Replication*—the data cache associated with single sign-on (SSO) sessions can be replicated to other security services in the cluster. This ensures that SSO session data is not lost if one member of the cluster should become unavailable.

This subsection describes how to configure failover and replication by hand.

Failover scenario

Example 30 shows a scenario for a highly available Orbix security service that consists of a cluster of three security services, each with an embedded login service. The security and login services run on separate hosts, `security01`, `security02`, and `security03` respectively, and all of the services rely on the same third-party LDAP database to store their user data.

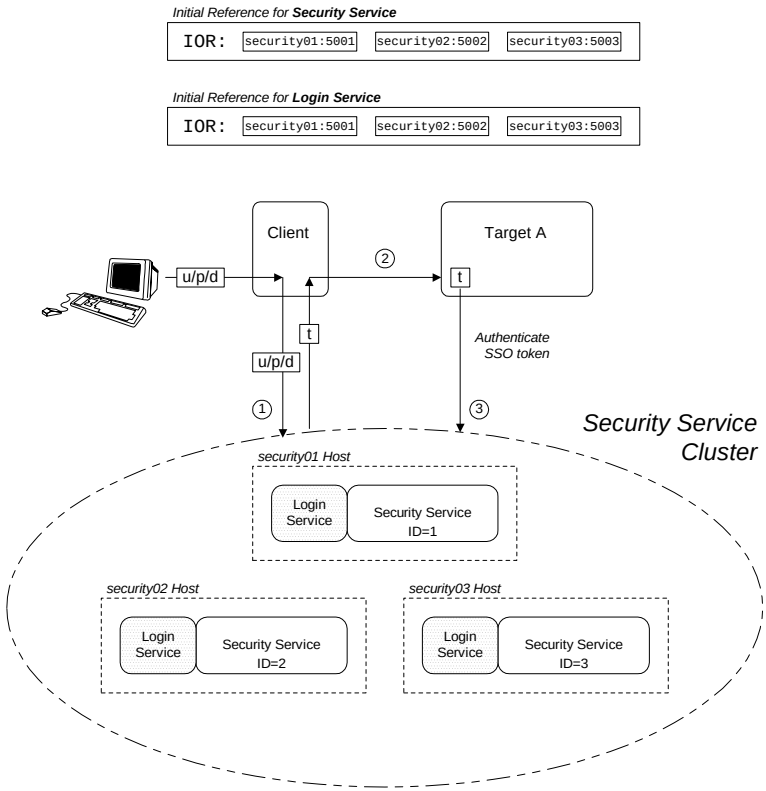

Figure 30: *Failover Scenario for a Cluster of Three Security Services*

In this scenario, it is assumed that both the client and the target application are configured to perform *random load balancing* over the security services in the cluster (see "Client Load Balancing" on page 165 for details). Each of the security services in the cluster are configured for failover and replication.

Failover scenario steps

The interaction of the client and target with the security service cluster shown in Example 30 on page 157 can be described as follows:

Stage	Description
1	Assuming the client is configured to use single sign-on (SSO), it will automatically contact the login service (which is part of the security service) to obtain an SSO token. Because the client is configured to perform random load balancing, it chooses one of the addresses from the `IT_Login` IOR at random and opens a connection to that login service.
2	The client invokes an operation on the target, sending the SSO token obtained in the previous step with the request.
3	The target server checks the SSO token received from the client by sending an invocation to the security service cluster. If the target server already has an existing connection with a service in the cluster, it re-uses that connection. Otherwise, the target randomly picks an address from the list of addresses in the `IT_SecurityService` IOR.

Configuring the is2.properties file

Each instance of the Orbix security service should have its own `is2.properties` file. Within each `is2.properties` file, you should set the following:

- `is2.current.server.id`—a unique ID for this Orbix security service instance,
- `is2.cluster.properties.filename`—a shared cluster file.
- `is2.replication.required`—must be set to `true`.
- `is2.replica.selector.classname`—you must set this variable as shown in the example.

For example, the first Orbix security server instance from Figure 30 on page 157 could be configured as follows:

```
# iS2 Properties File, for Server ID=1
...
##############################################
## iSF federation related properties
##############################################
is2.current.server.id=1
is2.cluster.properties.filename=C:/is2_config/cluster.properties
is2.replication.required=true
is2.replication.interval=20
is2.replica.selector.classname=com.iona.security.replicate.Stati
   cReplicaSelector
...
```

The second and third Orbix security services from Figure 30 on page 157 should be configured similarly, except that the `is2.current.server.id` property should be set to `2` and `3` respectively.

Configuring the cluster properties file

For the three-service cluster shown in Figure 30 on page 157, you could configure the `cluster.properties` file as follows:

```
# Advertise the locations of the security services in the cluster.
com.iona.security.common.securityInstanceURL.1=corbaloc:it_iiops:1.2@security01:5001/IT_Security
   Service
com.iona.security.common.securityInstanceURL.2=corbaloc:it_iiops:1.2@security02:5002/IT_Security
   Service
com.iona.security.common.securityInstanceURL.3=corbaloc:it_iiops:1.2@security03:5003/IT_Security
   Service

# Configure replication between security services.
com.iona.security.common.replicaURL.1=corbaloc:it_iiops:1.2@security02:5002/IT_SecurityService,c
   orbaloc:it_iiops:1.2@security03:5003/IT_SecurityService
com.iona.security.common.replicaURL.2=corbaloc:it_iiops:1.2@security03:5003/IT_SecurityService,c
   orbaloc:it_iiops:1.2@security01:5001/IT_SecurityService
com.iona.security.common.replicaURL.3=corbaloc:it_iiops:1.2@security01:5001/IT_SecurityService,c
   orbaloc:it_iiops:1.2@security02:5002/IT_SecurityService
```

There are two groups of settings in this file:

- `securityInstanceURL.`*ServerID*—advertises the location of a security service in the cluster. Normally, the most convenient way to set these values is to use the `corbaloc` URL format.

- `replicaURL.`*ServerID*—a list of URLs for the other security services to which this service replicates its data.

  For example, the `replicaURL.1` setting lists URLs for the security service with ID=2 and the security service with ID=3. Hence, the first service in the cluster is configured to replicate its data to the second and third services. Normally, each security service should replicate to all of the other services in the cluster.

Orbix configuration for the first security service

Example 18 shows the details of the Orbix configuration for the first Orbix security service in the cluster. To configure this security service to support failover, you must ensure that the security service's IOR contains a list addresses for all of the services in the cluster.

Example 18: *Orbix Security Service Configuration for Failover*

```
# Orbix Configuration File
1  initial_references:IT_SecurityService:reference =
      "IOR:010000002400000049444c3a696f6e612e636f6d2f49545f53656375
      726974792f5365727665723a312e3000010000000000000092000000001010
      2000800000066626f6c74616e0000000000220000003a3e02333109536563
      7572697479001249545f536563757269747953657276696365000004000000
      0140000000800000001007e005e0078cf0000000080000000100000415f
      5449010000001c0000000100000001000100010000000100010509010100
      1000000000101000600000006000000010000000e00";
   initial_references:IT_Login:reference =
      "IOR:010000002300000049444c3a696f6e612e636f6d2f49545f53656375
      726974792f4c6f67696e3a312e3000000100000000000000860000000010
      2000800000066626f6c74616e0000000000180000003a3e02333109536563
      7572697479000849545f4c6f67696e040000001400000008000000001007e0
      01e0078cf0000000080000000100000415f5449010000001c0000000100
      0000010001000100000001000105090101000100000000010100060000000
      6000000010000000e00";

   iona_services {
       ...
2      principal_sponsor:use_principal_sponsor = "true";
       principal_sponsor:auth_method_id = "pkcs12_file";
      principal_sponsor:auth_method_data = ["filename=PKCS12File",
      "password_file=CertPasswordFile"];

       policies:client_secure_invocation_policy:requires =
      ["Confidentiality", "EstablishTrustInTarget",
      "DetectMisordering", "DetectReplay", "Integrity"];
```

Example 18: *Orbix Security Service Configuration for Failover*

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

    security {
        Hostname {
            ...
3           plugins:security_cluster:iiop_tls:addr_list =
   ["+security01:5001", "+security02:5002", "+security03:5003"];
4           plugins:security:iiop_tls:host = "5001";
            plugins:security:iiop_tls:port = "security01";

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


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

The preceding Orbix configuration can be explained as follows:

1. The `IT_SecurityService` initial reference is read by Orbix applications to locate the cluster of Orbix security services. Embedded in this IOR is a list of addresses for all of the security services in the cluster.

   This IOR is generated by the Orbix security service when it is run in `prepare` mode—alternatively, you can use the `itconfigure` utility to create a domain with an Orbix security service cluster.

   Note: You can parse the contents of the stringified IOR using the `iordump` tool.

2. The Orbix security service picks up most of its SSL/TLS security settings from the `iona_services` scope. In particular, the default configuration of the security service uses the X.509 certificate specified by the `principal_sponsor` settings in this scope.
3. The `plugins:security_cluster:iiop_tls:addr_list` variable lists the addresses for all of the security services in the cluster. Each address in the list is preceded by a + sign, which indicates that the service embeds the address in its generated IORs.

   Note: The `plugins:security_cluster:iiop_tls:addr_list` setting also configures the embedded login service.

4. The `plugins:security:iiop_tls:host` and `plugins:security:iiop_tls:port` settings specify the address where the security service listens for incoming IIOP/TLS request messages.

Orbix configuration for other services in the cluster

The configuration for other services in the cluster is similar, except that the `plugins:security:iiop_tls:host` and `plugins:security:iiop_tls:port` variables should be changed to the appropriate host and port for each of the replicas.

Replication

Example 31 on page 163 shows how replication works in a cluster of three Orbix security services. If replication is enabled (that is, `is2.replication.required` is set to `true` in the `is2.properties` file), a security service pushes its data cache to the other services in the cluster every 30 seconds (default replication interval).

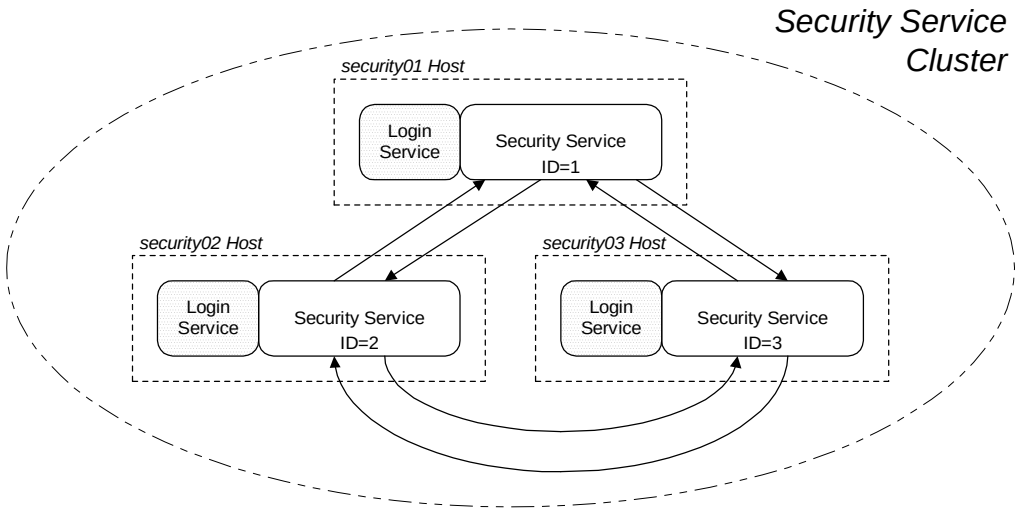

Figure 31: *Replication of Data Caches in a Security Service Cluster*

Security service replication has the following characteristics:

- The security service pushes the following data to the other services:
  - SSO tokens that have been added since the last replication.
  - Realm and role data for each of the new SSO tokens.
- Note, however, that the security service does *not* replicate username and password data. Therefore, replication is only relevant to applications that use the SSO feature.

Modifying the replication interval

You can modify the replication interval by setting the `is2.replication.interval` property in the `is.properties` file for the relevant service. If this variable is not set, the default replication interval is 30 seconds.

For example, to configure the security service with ID=1 to replicate data once every 10 seconds, its `is2.properties` file would be configured as follows:

```
# iS2 Properties File, for Server ID=1
...
##############################################
## iSF federation related properties
##############################################
is2.current.server.id=1
is2.cluster.properties.filename=C:/is2_config/cluster.properties
is2.replication.required=true
is2.replication.interval=10
is2.replica.selector.classname=com.iona.security.replicate.Stati
   cReplicaSelector
...
```

Client Load Balancing

Overview

When you use a clustered security service, it is important to configure *all* of the other applications in the system (clients and servers) to perform *client load balancing* (in this context, *client* means a client of the Orbix security service and thus includes ordinary Orbix servers as well). This ensures that the client load is evenly spread over all of the security services in the cluster.

Client load balancing is enabled by default.

Configuration for load balancing

Example 19 shows an outline of the configuration for a client of a security service cluster. Such clients must be configured to use random load balancing to ensure that the load is spread evenly over the servers in the cluster. The settings highlighted in bold should be added to the application's configuration scope.

Example 19: *Configuration for Client of a Security Service Cluster*

```
# Orbix Configuration File
...
load_balanced_app {
    ...
    plugins:gsp:use_client_load_balancing = "true";
    policies:iiop_tls:load_balancing_mechanism = "random";
};
```

Client load balancing mechanism

The client load balancing mechanism is selected by setting the `policies:iiop_tls:load_balancing_mechanism` variable. Two mechanisms are supported, as follows:

- `random`—choose one of the addresses embedded in the IOR at random (this is the default).

  Note: This is the only mechanism suitable for use in a deployed system.

- `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.

  In general, this mechanism is not recommended for deployed systems, because it usually results in all of the client applications connecting to the first cluster member.

Additional Security Configuration

Overview

This section describes how to configure optional features of the Orbix security service, such as single sign-on and the authorization manager. These features can be combined with any iSF adapter type.

In this section

This section contains the following subsections:

Configuring Single Sign-On Properties

Overview

The Orbix Security Framework provides an optional *single sign-on* (SSO) feature. If you want to use SSO with your applications, you must configure the Orbix security service as described in this section. SSO offers the following advantages:

- User credentials can easily be propagated between applications in the form of an SSO token.
- Performance is optimized, because the authentication step only needs to be performed once within a distributed system.
- Because the user's session is tracked centrally by the Orbix security service, it is possible to impose timeouts on the user sessions and these timeouts are effective throughout the distributed system.

SSO tokens

The login service generates an SSO token in response to an authentication operation. The SSO token is a compact key that the Orbix security service uses to access a user's session details, which are stored in a cache.

SSO properties

Example 20 shows the iSF properties needed for SSO:

Example 20: *Single Sign-On Properties*

```
# iSF Properties File
...
##############################################
## Single Sign On Session Info
##############################################
1 is2.sso.enabled=yes
2 is2.sso.session.timeout=6000
3 is2.sso.session.idle.timeout=300
4 is2.sso.cache.size=10000
```

The SSO properties are described as follows:

1. Setting this property to yes enables single sign-on.
2. The SSO session timeout sets the lifespan of SSO tokens, in units of seconds. Once the specified time interval elapses, the token expires.

3. The SSO session idle timeout sets the maximum length of time for which an SSO session can remain idle, in units of seconds. If the Orbix security service registers no activity against a particular session for this amount of time, the session and its token expire.
4. The size of the SSO cache, in units of number of sessions.

Related administration tasks

For details of how to configure CORBA applications to use SSO, see "Single Sign-On for CORBA Applications" on page 99.

Configuring the Log4J Logging

Overview

log4j is a third-party toolkit from the Jakarta project, http://jakarta.apache.org/log4j, that provides a flexible and efficient system for capturing logging messages from an application. Because the Orbix security service's logging is based on log4j, it is possible to configure the output of Orbix security service logging using a standard log4j properties file.

log4j documentation

For complete log4j documentation, see the following Web page:

http://jakarta.apache.org/log4j/docs/documentation.html

Enabling log4j logging

To enable log4j logging, you can specify the location of the log4j properties file in either of the following ways:

- In the system_properties list.
- In the SECURITY_CLASSPATH.

In the system_properties list

You can specify the location of the log4j properties file by setting the `com.iona.common.log4j.Log4JUtils.filename` property in the `plugins:java_server:system_properties` list in the security service configuration. For example, to use the `/is2_config/log4j.properties` file, modify the security service configuration by extending its system properties list as follows:

```
# Orbix Configuration File
# In the security service configuration scope:
plugins:java_server:system_properties = [...,
   "com.iona.common.log4j.Log4JUtils.filename=/is2_config/log4j.
   properties"];
```

In the SECURITY_CLASSPATH

You can specify the location of the log4j properties file by adding it to the `SECURITY_CLASSPATH` variable in the Orbix configuration file (the separator between items in the classpath is `;` on Windows platforms and `:` on UNIX platforms).

Configuring the log4j properties file

The following example shows how to configure the log4j properties to perform basic logging. In this example, the lowest level of logging is switched on (DEBUG) and the output is sent to the console screen.

```
# log4j Properties File
log4j.rootCategory=DEBUG, A1

# A1 is set to be a ConsoleAppender.
log4j.appender.A1=org.apache.log4j.ConsoleAppender

# A1 uses PatternLayout.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%-4r [%t] %-5p %c %x
   - %m%n
```

CHAPTER 7

Managing Users, Roles and Domains

The Orbix security service provides a variety of adapters that enable you to integrate the IONA Security Framework with third-party enterprise security products. This allows you to manage users and roles using a third-party enterprise security product.

In this chapter

This chapter discusses the following topics:

Introduction to Domains and Realms

Overview

This section introduces the concepts of an iSF security domain and an iSF authorization realm, which are fundamental to the administration of the IONA Security Framework. Within an iSF security domain, you can create user accounts and within an iSF authorization realm you can assign roles to users.

In this section

This section contains the following subsections:

iSF Security Domains

Overview

This subsection introduces the concept of an iSF security domain.

iSF security domain

An *iSF security domain* is a particular security system, or namespace within a security system, designated to authenticate a user.

Here are some specific examples of iSF security domains:

- LDAP security domain—authentication provided by an LDAP security backend, accessed through the Orbix security service.

Domain architecture

Figure 32 shows the architecture of an iSF security domain. The iSF security domain is identified with an enterprise security service that plugs into the Orbix security service through an iSF adapter. User data needed for authentication, such as username and password, are stored within the enterprise security service. The Orbix security service provides a central access point to enable authentication within the iSF security domain.

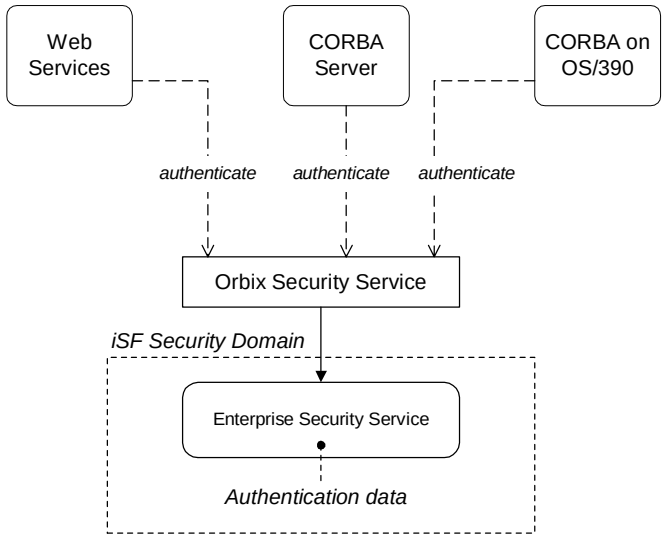

Figure 32: *Architecture of an iSF Security Domain*

Creating an iSF security domain

Effectively, you create an iSF security domain by configuring the Orbix security service to link to an enterprise security service through an iSF adapter (such as an LDAP adapter). The enterprise security service is the implementation of the iSF security domain.

Creating a user account

Because user account data is stored in a third-party enterprise security service, you use the standard tools from the third-party enterprise security product to create a user account.

For a simple example, see "Managing a File Security Domain" on page 187.

iSF Authorization Realms

Overview

This subsection introduces the concept of an iSF authorization realm and role-based access control, explaining how users, roles, realms, and servers are interrelated.

iSF authorization realm

An *iSF authorization realm* is a collection of secured resources that share a common interpretation of role names. An authenticated user can have different roles in different realms. When using a resource in realm `R`, only the user's roles in realm `R` are applied to authorization decisions.

Role-based access control

The IONA security framework supports a *role-based access control* (RBAC) authorization scheme. Under RBAC, authorization is a two step process, as follows:

1. User-to-role mapping—every user is associated with a set of roles in each realm (for example, `guest`, `administrator`, and so on, in a realm, `Engineering`). A user can belong to many different realms, having a different set of roles in each realm.

   The user-to-role assignments are managed centrally by the Orbix security service, which returns the set of realms and roles assigned to a user when required.
2. Role-to-permission mapping (or action-role mapping)—in the RBAC model, permissions are granted to *roles*, rather than directly to users. The role-to-permission mapping is performed locally by a server, using data stored in local access control list (ACL) files. For example, CORBA servers in the iSF use an XML action-role mapping file to control access to IDL interfaces, operation, and attributes.

Servers and realms

From a server's perspective, an iSF authorization realm is a way of grouping servers with similar authorization requirements. Figure 33 shows two iSF authorization realms, `Engineering` and `Finance`, each containing a collection of server applications.

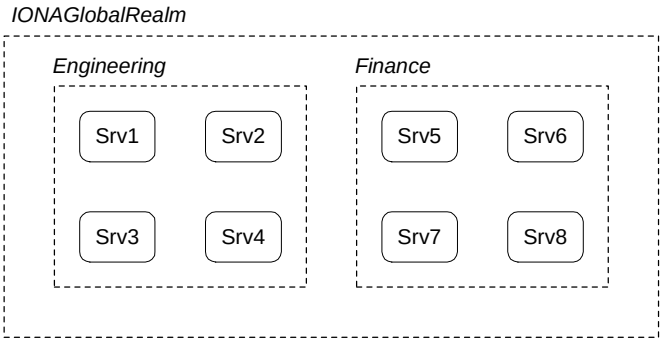

Figure 33: *Server View of iSF Authorization Realms*

Adding a server to a realm

To add a server to a realm, add or modify the `plugins:gsp:authorization_realm` configuration variable within the server's configuration scope (either in the *DomainName*`.cfg` file or in the CFR server).

For example, if your server's configuration is defined in the `my_server_scope` scope, you can set the iSF authorization realm to `Engineering` as follows:

```
# Orbix configuration file
...
my_server_scope {
    plugins:gsp:authorization_realm = "Engineering";
    ...
};
```

Roles and realms

From the perspective of role-based authorization, an iSF authorization realm acts as a namespace for roles. For example, Figure 34 shows two iSF authorization realms, `Engineering` and `Finance`, each associated with a set of roles.

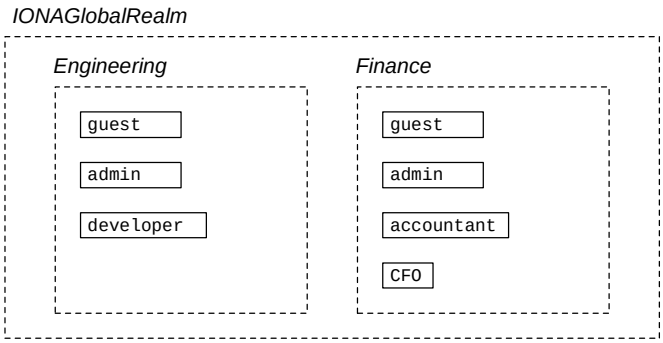

Figure 34: *Role View of iSF Authorization Realms*

Creating realms and roles

Realms and roles are usually administered from within the enterprise security system that is plugged into the Orbix security service through an adapter. Not every enterprise security system supports realms and roles, however.

For example, in the case of a security file connected to a file adapter (a demonstration adapter provided by IONA), a realm or role is implicitly created whenever it is listed amongst a user's realms or roles. See also "Assigning realms and roles to the example users" on page 181.

Assigning realms and roles to users

The assignment of realms and roles to users is administered from within the enterprise security system that is plugged into the Orbix security service. For example, Figure 35 shows how two users, `Janet` and `John`, are assigned roles within the `Engineering` and `Finance` realms.

- Janet works in the engineering department as a developer, but occasionally logs on to the `Finance` realm with guest permissions.

- John works as an accountant in finance, but also has guest permissions with the `Engineering` realm.

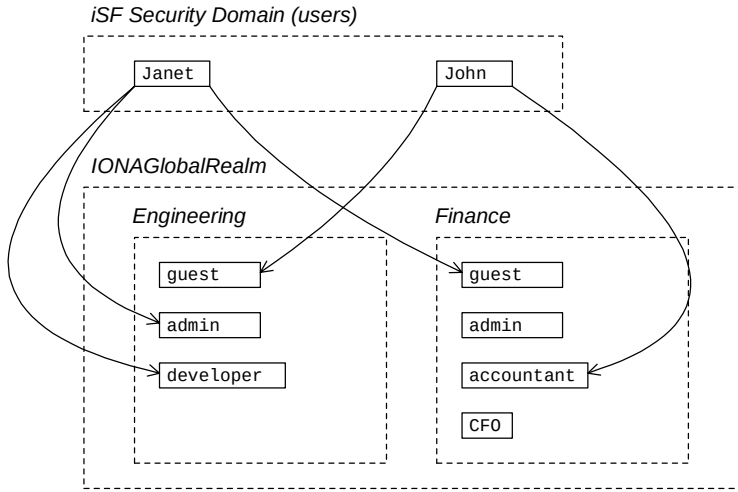

Figure 35: *Assignment of Realms and Roles to Users Janet and John*

Special realms and roles

The following special realms and roles are supported by the IONA Security Framework:

- `IONAGlobalRealm` realm—a special realm that encompasses every iSF authorization realm. Roles defined within the `IONAGlobalRealm` are valid within every iSF authorization realm.
- `UnauthenticatedUserRole`—a special role that can be used to specify actions accessible to an unauthenticated user (in an action-role mapping file). An unauthenticated user is a remote user without credentials (that is, where the client is not configured to send GSSUP credentials).

  Actions mapped to the `UnauthenticatedUserRole` role are also accessible to authenticated users.

  The `UnauthenticatedUserRole` can be used *only* in action-role mapping files.

Example Domain and Realms

Overview

This subsection presents an example of how to set up an iSF security domain using a file domain. Sample iSF authorization realms, roles, and users are created, and the authorization process is explained by example.

File domain

In this example, the iSF security domain is configured to be a *file domain*. A file domain is a simple file-based security domain that can be used for tests or demonstrations. The user data is then stored in an XML security file.

For details of how to configure a file domain, see "Managing a File Security Domain" on page 187.

Example users

The following users are created in the file domain for this example:

- Janet—with username, `Janet`, and password, `JanetPass`.
- John—with username, `John`, and password, `JohnPass`.
- SuperUser—with username, `SuperUser`, and password, `BigSecret`.

Assigning realms and roles to the example users

The following realms and roles are assigned to the users, `Janet`, `John`, and `SuperUser` (where realms and roles are notated in the format *RealmA* `{` *roleA1, roleA2, ..., roleAn*`}`):

- `Janet`—is assigned the following realms and roles:
  - `Engineering {developer, admin}`
  - `IONAGlobalRealm {guest}`
- `John`—is assigned the following realms and roles:
  - `Finance {accountant}`
  - `IONAGlobalRealm {guest}`
- `SuperUser`—is assigned the following realm and role:
  - `IONAGlobalRealm {admin}`

Sample security file for the file domain

Within a file domain, you specify the user authentication data (username and password) as well as the realm/role assignments within the same XML security file. The preceding user data can be specified in a security file as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<ns:securityInfo xmlns:ns="urn:www-xmlbus-com:simple-security">
  <users>
    <user name="Janet" password="JanetPass"
          description="Developer">
      <realm name="Engineering">
        <role name="developer"/>
        <role name="admin"/>
      </realm>
      <realm name="IONAGlobalRealm" description="All realms">
        <role name="guest"/>
      </realm>
    </user>
    <user name="John" password="JohnPass"
          description="Accountant">
      <realm name="Finance">
        <role name="accountant"/>
      </realm>
      <realm name="IONAGlobalRealm" description="All realms">
        <role name="guest"/>
      </realm>
    </user>
    <user name="SuperUser" password="BigSecret"
          description="All powerful user!">
      <realm name="IONAGlobalRealm" description="All realms">
        <role name="admin" description="All actions"/>
      </realm>
    </user>
  </users>
</ns:securityInfo>
```

Sample server configuration

Consider, for example, the CORBA naming service in the `Engineering` iSF authorization realm. To configure this naming service, edit the variables in the `iona_services.naming` scope in the *DomainName*`.cfg` configuration file. Set the authorization realm to `Engineering` and specify the location of the action-role mapping file, as follows:

```
# Orbix configuration file
...
iona_services {
    ...
    naming {
        plugins:gsp:authorization_realm = "Engineering";
        plugins:is2_authorization:action_role_mapping =
            "file:///security/eng_naming_arm.xml";
        ...
    };
};
```

Sample ACL file

The `eng_naming_arm.xml` action-role mapping file, which specifies permissions for the naming service in the `Engineering` domain, could be defined as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE secure-system SYSTEM
   "actionrolemapping_with_interfaces.dtd">
<secure-system>
  <allow-unlisted-interfaces>true</allow-unlisted-interfaces>
  <action-role-mapping>
    <server-name>iona_services.naming</server-name>
    <interface>
      <name>IDL:omg.org/CosNaming/NamingContext:1.0</name>
      <action-role>
        <action-name>*</action-name>
        <role-name>developer</role-name>
      </action-role>
      <action-role>
        <action-name>resolve</action-name>
        <action-name>list</action-name>
        <role-name>guest</role-name>
      </action-role>
    </interface>
  </action-role-mapping>
</secure-system>
```

Authorization process

When user John attempts to invoke an operation on the CORBA naming service in the `Engineering` domain, authorization proceeds as follows:

Stage	Description
1	The naming service contacts the Orbix security service remotely to authenticate John's username and password.
2	If authentication is successful, the Orbix security service returns the complete list of realms and roles assigned to John. In the current example, the following realms and roles would be returned: • `Finance {accountant}` • `IONAGlobalRealm {guest}`
3	The naming service determines which roles are applicable to John in the current iSF authorization realm. Because the naming service belongs to the `Engineering` realm, only the `guest` role from the `IONAGlobalRealm` is applicable here.
4	The naming service now checks the `eng_naming_arm.xml` action-role mapping file and finds that only the `resolve` and `list` actions are permitted on the `CosNaming::NamingContext` IDL interface for the `guest` role. On the other hand, if the user, John, attempts to call an operation (or attribute) on any other naming service interface, the call would be permitted, because the `<allow-unlisted-interfaces>` option is `true` in the action-role mapping file. **Note:** The special `<allow-unlisted-interfaces>` tag is a useful shortcut, but you should use it carefully to avoid opening a security hole.

Domain and Realm Terminology

Overview

The terms *domain* and *realm* appear in several security technology specifications with different (and sometimes contradictory) meanings. This subsection attempts to clarify some of the domain and realm terminology and provides a comparison with the IONA Security Framework terms.

Comparison of terminology

To clarify the terminology used by different technology specifications (all of which are embraced by the iSF) Table 4 lists the generic iSF terms against their technology-specific equivalents:

Table 4: *Domain and Realm Terminology Comparison*

Generic iSF Term	Technology-Specific Equivalents
iSF security domain	J2EE security technology domain J2EE security policy domain (**1**) J2EE realm (**2**) JAAS authentication realm CSIv2 authentication domain HTTP login realm
iSF authorization realm	J2EE security policy domain (**1**) J2EE realm (**2**)

1. The term, J2EE security policy domain, appears in both rows because it is a general term that embodies both an authentication domain and an authorization domain.
2. J2EE realm means the same thing as J2EE security policy domain.

J2EE security technology domain

The J2EE specification defines a *J2EE security technology domain* as follows:

The scope over which a single security mechanism is used to enforce a security policy. Multiple security policy domains can exist within a single technology domain.

J2EE security policy domain	The J2EE specification defines a *J2EE security policy domain* as follows: A realm, also called a security policy domain or security domain in the J2EE specification, is a scope over which a common security policy is defined and enforced by the security administrator of the security service
J2EE realm	A *J2EE realm* is the same thing as J2EE security policy domain.
JAAS authentication realm	A Java Authentication and Authorization Service (JAAS) authentication realm is a namespace for JAAS principals.
CSIv2 authentication domain	A *CSIv2 authentication domain* is a named domain in which CSIv2 authentication data (for example, username and password) is authenticated.
HTTP login realm	When a user logs on to a Web client through a standard HTTP login mechanism (for example, HTTP basic authentication or HTTP form-based authentication), the user is prompted for a username, password, and login realm name. The login realm name, along with the user's username and password, is the sent to the Web server.

Managing a File Security Domain

Overview

The file security domain is active if the Orbix security service has been configured to use the iSF file adapter (see "Configuring the File Adapter" on page 142). The main purpose of the iSF file adapter is to provide a lightweight security domain for demonstration purposes. A realistic deployed system, however, would use one of the other adapters (LDAP or custom) instead.

WARNING: The file adapter is provided for demonstration purposes only. IONA does not support the use of the file adapter in a production environment.

Location of file

The location of the security information file is specified by the `com.iona.isp.adapter.file.param.filename` property in the Orbix security service's `is2.properties` file.

Example

Example 21 is an extract from a sample security information file that shows you how to define users, realms, and roles in a file security domain.

Example 21: *Sample Security Information File for an iSF File Domain*

```
<?xml version="1.0" encoding="utf-8" ?>

<ns:securityInfo xmlns:ns="urn:www-xmlbus-com:simple-security">
  <users>
    <user name="IONAAdmin" password="admin"
          description="Default IONA admin user">
      <realm name="IONA" description="All IONA applications"/>
    </user>
    <user name="admin" password="admin" description="Old admin
   user; will not have the same default privileges as
   IONAAdmin.">
      <realm name="Corporate">
        <role name="Administrator"/>
      </realm>
    </user>
    <user name="alice" password="dost1234">
```

Example 21: *Sample Security Information File for an iSF File Domain*

```
5       <realm name="Financials"
                description="Financial Department">
          <role name="Manager" description="Department Manager" />
          <role name="Clerk"/>
        </realm>
      </user>
      <user name="bob" password="dost1234">
        <realm name="Financials">
          <role name="Clerk"/>
        </realm>
      </user>
    </users>
</ns:securityInfo>
```

1. The <ns:securityInfo> tag can contain a nested <users> tag.
2. The <users> tag contains a sequence of <user> tags.
3. Each <user> tag defines a single user. The <user> tag's name and password attributes specify the user's username and password. Within the scope of the <user> tag, you can list the realms and roles with which the user is associated.
4. When a <realm> tag appears within the scope of a <user> tag, it implicitly defines a realm and specifies that the user belongs to this realm. A <realm> must have a name and can optionally have a description attribute.
5. A realm can optionally be associated with one or more roles by including <role> elements within the <realm> scope.

Certificate-based authentication for the file adapter

When performing certificate-based authentication, the file adapter compares the certificate to be authenticated with a cached copy of the user's certificate.

To configure the file adapter to support X.509 certificate-based authentication, perform the following steps:

1. Cache a copy of each user's certificate, *CertFile*.pem, in a location that is accessible to the file adapter.
2. Make the following type of entry for each user with a certificate:

   Example 22: *File Adapter Entry for Certificate-Based Authentication*

   ```
   ...
   <user name="CNfromSubjectDN" certificate="CertFile.pem"
      description="User certificate">
     <realm name="RealmName">
       ...
     </realm>
   </user>
   ```

   The user's name, *CNfromSubjectDN*, is derived from the certificate by taking the Common Name (CN) from the subject DN of the X.509 certificate (for DN terminology, see "ASN.1 and Distinguished Names" on page 629). The `certificate` attribute specifies the location of this user's X.509 certificate, *CertFile*`.pem`.

Managing an LDAP Security Domain

Overview

The Lightweight Directory Access Protocol (LDAP) can serve as the basis of a database that stores users, groups, and roles. There are many implementations of LDAP and any of them can be integrated with the Orbix security service by configuring the LDAP adapter.

Please consult documentation from your third-party LDAP implementation for detailed instructions on how to administer users and roles within LDAP.

Configuring the LDAP adapter

A prerequisite for using LDAP within the IONA Security Framework is that the Orbix security service be configured to use the LDAP adapter.

See "Configuring the LDAP Adapter" on page 144.

Certificate-based authentication for the LDAP adapter

When performing certificate-based authentication, the LDAP adapter compares the certificate to be authenticated with a cached copy of the user's certificate.

To configure the LDAP adapter to support X.509 certificate-based authentication, perform the following steps:

1. Cache a copy of each user's certificate, *CertFile*.pem, in a location that is accessible to the LDAP adapter.
2. The user's name, *CNfromSubjectDN*, is derived from the certificate by taking the Common Name (CN) from the subject DN of the X.509 certificate (for DN terminology, see "ASN.1 and Distinguished Names" on page 629).
3. Make (or modify) an entry in your LDAP database with the username, *CNfromSubjectDN*, and specify the location of the cached certificate.

CHAPTER 8

Managing Access Control Lists

The Orbix Security Framework defines access control lists (ACLs) for mapping roles to resources. The ACLs are specific to particular technology domains, such as CORBA, and thus are not stored centrally in the Orbix security service.

In this chapter

This chapter discusses the following topics:

CORBA ACLs

Overview

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

In this section

This section contains the following subsections:

Overview of CORBA ACL Files

Action-role mapping file

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

GSP plug-in

The GSP plug-in is a component of the iSF that provides support for action-role mapping. This plug-in must be loaded in order to use the action-role mapping ACL file (see "Security Configuration" on page 485 for details of how to configure the GSP plug-in).

CORBA Action-Role Mapping ACL

Overview

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

File location

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

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

Example IDL

For example, consider how to set the operation and attribute permissions for the IDL interface shown in Example 23.

Example 23: *Sample IDL for CORBA ACL Example*

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

Example action-role mapping

Example 24 shows how you might configure an action-role mapping file for the `Simple::SimpleObject` interface given in the preceding Example 23 on page 194.

Example 24: *CORBA Action-Role Mapping Example*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE secure-system SYSTEM
   "InstallDir/etc/domains/Domain/actionrolemapping.dtd">
<secure-system>
  <allow-unlisted-interfaces>false</allow-unlisted-interfaces>

  <action-role-mapping>
    <server-name>gsp_basic_test.server</server-name>
    <interface>
      <name>IDL:Simple/SimpleObject:1.0</name>
      <action-role>
        <action-name>call_me</action-name>
        <role-name>corba-developer</role-name>
        <role-name>guest</role-name>
      </action-role>
      <action-role>
        <action-name>_get_foo</action-name>
        <role-name>corba-developer</role-name>
        <role-name>guest</role-name>
      </action-role>
    </interface>

  </action-role-mapping>
</secure-system>
```

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

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

- `false`—for any interfaces not listed, access is denied for all roles. Unauthenticated users are also denied access. This is the default.

3. The `<action-role-mapping>` tag contains all of the permissions that apply to a particular server application.
4. The `<server-name>` tag specifies the ORB name that is used by the server in question. The value of this tag must match the ORB name exactly.

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

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

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

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

7. The `call_me` action name corresponds to the `call_me()` operation in the `Simple::SimpleObject` interface. The action name corresponds to the GIOP on-the-wire form of the operation name (usually the same as it appears in IDL).
8. The `_get_foo` action name corresponds to the `foo` attribute accessor. In general, any read/write attribute, *AttributeName*, has the following action names:
   - `_get_`*AttributeName*—for the attribute accessor, and
   - `_set_`*AttributeName*—for the attribute modifier.

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

Action-role mapping DTD

The syntax of the action-role mapping file is defined by the action-role mapping DTD. See "Action-Role Mapping DTD" on page 533 for details.

Centralized ACL

Overview

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

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

In this section

This section contains the following subsections:

Local ACL Scenario

Overview

This section briefly describes the behavior of a secure server whose operations are protected by a local ACL file (see, for example, "Target configuration" on page 81 for details of such a configuration).

Local ACL scenario

Figure 36 shows an outline of the local ACL scenario, where the ACL file is stored on the same host as the target server. You configure the server to load the ACL file from the local file system by setting the `plugins:gsp:action_role_mapping_file` variable in the target server's configuration scope.

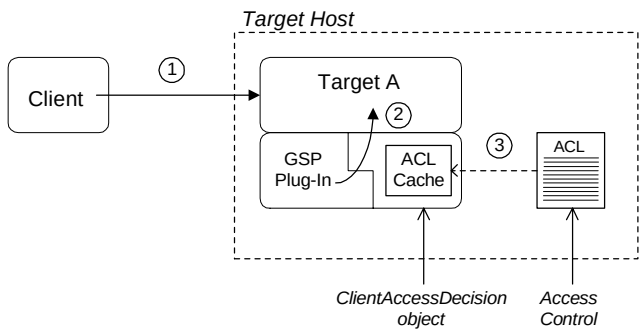

Figure 36: *Local ACL Scenario*

Scenario description

The local ACL scenario shown in Figure 36 can be described as follows:

Stage	Description
1	The client invokes an operation on the secure target server, requiring an access decision to be made on the server side.
2	The GSP plug-in calls a function on the internal `ClientAccessDecision` object to check whether the current user has permission to invoke the current operation.

Stage	Description
3	If this is the first access decision required by the target server, the `ClientAccessDecision` object reads the contents of the local ACL file (as specified by the `plugins:gsp:action_role_mapping_file` variable) and stores the ACL data in a cache. For all subsequent access decisions, the `ClientAccessDecision` object reads the cached ACL data for efficiency.

Centralized ACL Scenario

Overview

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

Centralized ACL scenario

Figure 37 shows an outline of a centralized ACL scenario, where the ACL files are stored on the same host as the Orbix security service.

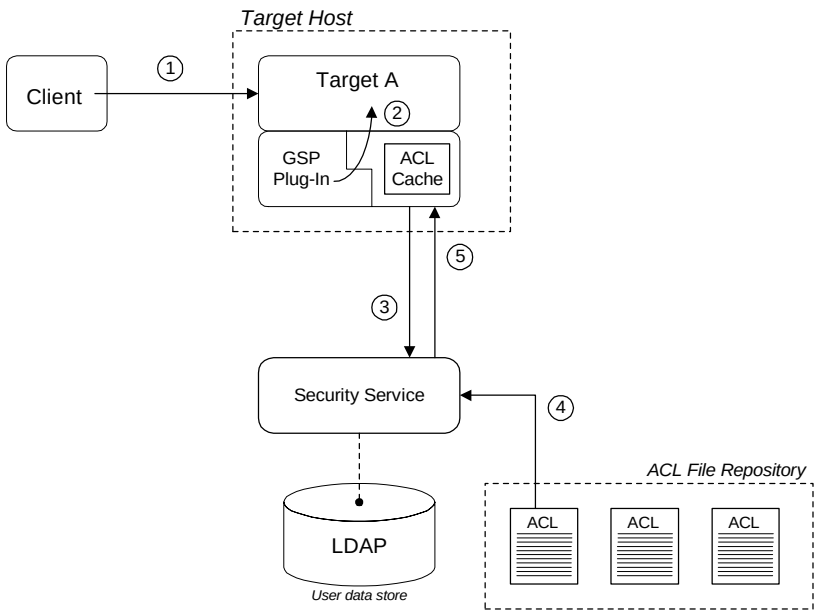

Figure 37: *Centralized ACL scenario*

Scenario description

The centralized ACL scenario shown in Figure 37 can be described as follows:

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

Modify the Orbix configuration file

To configure an application (such as the target server shown in Figure 37 on page 201) to use a centralized ACL, you must modify its configuration scope as shown in Example 25. In this example, it is assumed that the application's ORB name is `my_secure_apps.my_two_tier_target`.

Example 25: *Configuration of a Second-Tier Target Server in the iSF*

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

Example 25: *Configuration of a Second-Tier Target Server in the iSF*

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

The preceding Orbix configuration can be described as follows:

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

Modify the is2.properties file

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

```
# is2.properties File for the Orbix Security Service
...
com.iona.isp.authz.adapters=file
com.iona.isp.authz.adapter.file.class=com.iona.security.is2AzAda
   pter.multifile.MultiFileAzAdapter
com.iona.isp.authz.adapter.file.param.filelist=ACLFileListFile;
```

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

Create an ACL file list file

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

```
[ACLKey=]ACLFileName
```

A file name can optionally be preceded by an ACL key and an equals sign, *ACLKey*=, if you want to select the file by ACL key (see "Selection by ACL key" on page 206). The ACL file, *ACLFileName*, is specified using an absolute pathname in the local file format.

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

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

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

Selecting the ACL file

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

- Selection by ORB name.
- Selection by override value.
- Selection by ACL key.

Selection by ORB name

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

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

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

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

Selection by override value

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

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

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

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

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

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

Selection by ACL key

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

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

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

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

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

Customizing Access Control Locally

Overview

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

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

Custom ClientAccessDecision in an Orbix application

Figure 38 shows an outline of an ACL scenario, where the default `ClientAccessDecision` object is replaced by a customized implementation.

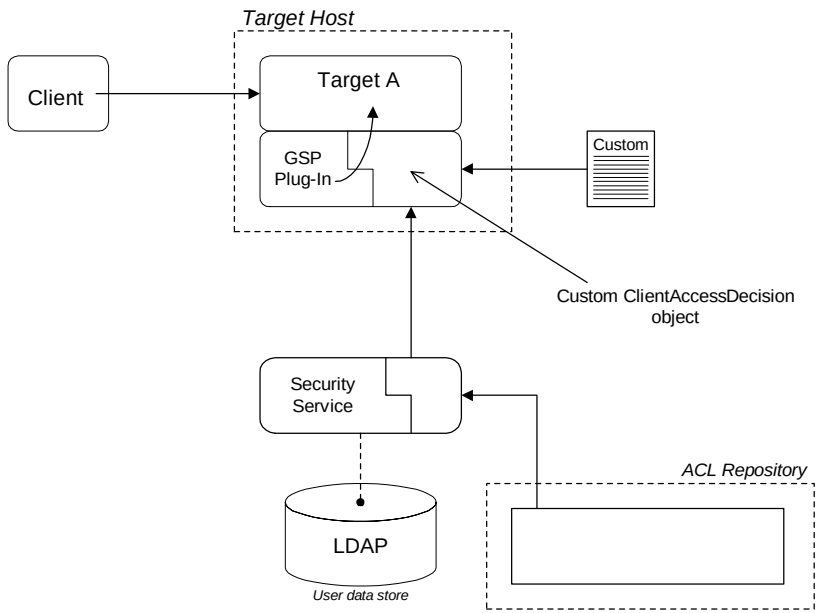

Figure 38: *Custom ClientAccessDecision in an Orbix Application*

Scenario variants

Replacing the `ClientAccessDecision` object with a customized implementation effectively gives you complete control over the access decision logic in an Orbix application. The system shown in Figure 38 can be adapted to a variety of scenarios, as follows:

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

CHAPTER 9

Securing Orbix Services

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

In this chapter

This chapter discusses the following topics:

Introduction to Securing Services

Overview

In a secure system, all Orbix services should be capable of servicing secure connections. A typical secure system includes an Orbix security service and enables SSL/TLS on all of the Orbix services.

Configuring the Orbix services

Before deploying the Orbix services in a live system, 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 63).

Configuring the Orbix security service

The Orbix security service is a special case because, in addition to setting configuration variables in the Orbix configuration, you also need to perform the following basic administration tasks:

- Edit the properties in the `is2.properties` file—see "Configuring the Orbix Security Service" on page 141.
- Change the secure user data (usernames, passwords, and so on) stored in the Orbix security service's user database—see "Managing Users, Roles and Domains" on page 173.

Access control lists for Orbix services

Fine-grained access to the Orbix services is controlled by the access control lists (ACLs) in the Orbix *action-role mapping* files. Default ACLs are generated automatically when you run `itconfigure` to create a secure domain. See "Default Access Control Lists" on page 227 for a detailed discussion of the default ACLs for the Orbix services.

File-Based and CFR Domains

Overview

This section provides an overview and comparison of a secure file-based domain and a secure CFR domain. There are some significant differences between the two types of domain. In particular, a secure CFR domain is designed in such a way as to avoid creating a circular dependency between the Orbix security service and the CFR service.

File-based domain overview

Figure 39 shows an overview of a secure file-based domain. In this example, the Orbix security service runs on a host, S1, and the other core Orbix services run on a different host, S2.

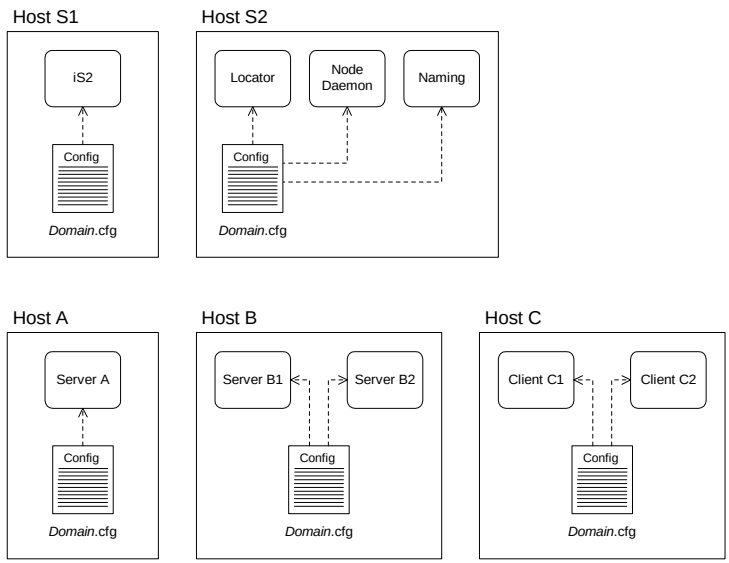

Figure 39: *Overview of a Secure File-Based Domain*

Domain.cfg in a file-based domain

In a secure file-based domain, the Orbix configuration file, *Domain*`.cfg`, contains all of the configuration data for the CORBA system. In particular, the *Domain*`.cfg` file can contain security credentials for your applications and the core Orbix services (for example, certificate locations and password file locations).

When deploying a domain across multiple hosts (as, for example, in Figure 39), it is advisable to customize the *Domain*`.cfg` file on each host. Each copy of *Domain*`.cfg` should include security credentials only for the applications running on that particular host.

WARNING: Any domain configuration files containing security-related data must be stored securely by the operating system.

CFR domain overview

Figure 40 shows an overview of a secure CFR domain. In this example, the Orbix security service runs on a host, S1, and the other core Orbix services run on a different host, S2.

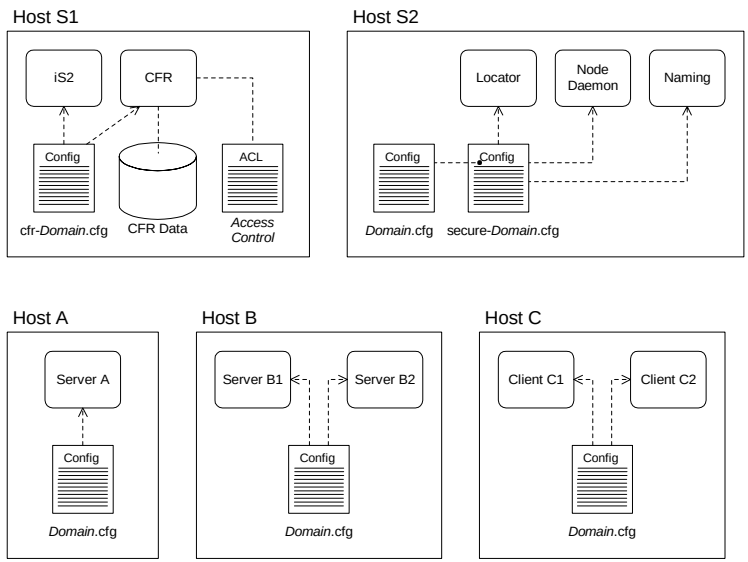

Figure 40: *Overview of a Secure CFR Domain*

Secure CFR domain files

A secure CFR domain uses the following different kinds of domain configuration files:

- Domain.cfg in a CFR domain.
- secure-Domain.cfg.
- cfr-Domain.cfg.

Domain.cfg in a CFR domain

In a secure CFR domain, the *Domain*`.cfg` file contains just enough configuration information to bootstrap an application and enable it to retrieve the rest of its configuration from the CFR service.The following kinds of settings are contained in this file:

- *Generic security settings*—for example, basic settings for the `iiop_tls` and `gsp` plug-ins.
- *CFR handler plug-in settings*—these settings tell the application to retrieve its configuration from the CFR service.
- *Credentials used by an internal ORB*—the internal ORB settings enable the Orbix management service to monitor the status of a server application.

secure-Domain.cfg

The `secure-`*Domain*`.cfg` file is used only by the core Orbix services (except the Orbix security service and the CFR service). It is generated only if the CFR and the Orbix security service are both deployed. Hence, in Figure 40 on page 212, the `secure-`*Domain*`.cfg` file appears only on the host where the Orbix services are deployed. The `secure-`*Domain*`.cfg` contains the following:

- All of the settings in *Domain*`.cfg`—the contents of the *Domain*`.cfg` are included using an `include` directive.
- Credentials for the core services—this includes credentials set by the IIOP/TLS principal sponsor and the CSIv2 principal sponsor.

WARNING: The `secure-`*Domain*`.cfg` file contains sensitive data and therefore it must be stored securely by the operating system.

cfr-Domain.cfg

The `cfr-`*Domain*`.cfg` file is used only by the Orbix security service and the CFR service (see Figure 40 on page 212) and it contains the *complete* configuration details for these two services. It is necessary to leave the configuration of these two services entirely file-based in order to avoid creating a circular dependency.

In a typical deployment, you need to customize the credentials for the Orbix security service and the CFR service, which are set in `cfr-`*Domain*`.cfg`.

WARNING: The `cfr-`*Domain*`.cfg` file contains sensitive data and therefore it must be stored securely by the operating system.

CFR action-role mapping

Like any of the other Orbix services, in a secure or semi-secure domain the CFR has an associated action-role mapping file. It is usually necessary to customize this action-role mapping in order to define which configuration scopes are accessible to ordinary users and which configuration scopes are reserved for the administrator.

For more details, see "Configuration Repository ACL" on page 228.

Customizing a Secure Domain

Overview

This section describes how to customize the configuration of secure domains generated using the `itconfigure` utility. When generating a domain, the `itconfigure` utility allows you to choose between two different levels of security:

- *Secure*—only secure connections are accepted.
- *Semi-secure*—both secure and insecure connections are accepted.

In the subsections that follow, the differences between a secure domain and a semi-secure domain are described in detail.

The `itconfigure` utility also allows you to choose between a file-based domain and a CFR-based domain. The examples in this section are all based on a file domain. Similar comments apply, though, to the analogous settings in a CFR domain.

WARNING: It is essential to customize a secure domain generated by the `itconfigure` utility. The secure domain created using `itconfigure` is *not* fully secure, because the X.509 certificates used by the domain are demonstration certificates, which are identical for all installations of Orbix.

In this section

This section contains the following subsections:

Configuring a Typical Orbix Service

Overview

This section describes how to configure a typical Orbix service—such as naming, trading, events, and so on—running in a domain with an Orbix security service. Details of the Orbix security service configuration are discussed in the next subsection "Configuring the Security Service" on page 224.

To configure a typical Orbix service, there are two groups of configuration settings that are relevant:

- Configuration settings for the application ORB—these settings configure the behavior of Orbix at the application level.
- Configuration settings for the internal ORB—these settings configure an internal ORB that allows the server process to be monitored by the Orbix management service.

Configuration settings for the application ORB

Example 26 shows the configuration settings for a typical Orbix service (not the security service itself). These settings configure the application ORB—that is, these settings determine the ordinary runtime behavior of the service.

Example 26: *Typical Service Configuration for the Application ORB*

```
# Orbix Configuration File
...
# General configuration at root scope.
1 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"];

2 policies:mechanism_policy:protocol_version = "SSL_V3";
  policies:mechanism_policy:ciphersuites =
    ["RSA_WITH_RC4_128_SHA", "RSA_WITH_RC4_128_MD5"];

3 policies:trusted_ca_list_policy =
    "/vob/art/etc/tls/x509/trusted_ca_lists/ca_list1.pem";
...
```

Example 26: *Typical Service Configuration for the Application ORB*

```
iona_services
{
    # Common SSL/TLS security settings.
4    principal_sponsor:use_principal_sponsor = "true";
5    principal_sponsor:auth_method_id = "pkcs12_file";
6    principal_sponsor:auth_method_data =
   ["filename=/vob/art/etc/tls/x509/certs/services/administrator
   .p12",
   "password_file=/vob/art/etc/tls/x509/certs/services/administr
   ator.pwf"];

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

8     policies:client_secure_invocation_policy:requires =
   ["Confidentiality", "EstablishTrustInTarget",
   "DetectMisordering", "DetectReplay", "Integrity"];
     policies:client_secure_invocation_policy:supports =
   ["Confidentiality", "EstablishTrustInClient",
   "EstablishTrustInTarget", "DetectMisordering",
   "DetectReplay", "Integrity"];

9    binding:server_binding_list = ["CSI+GSP+OTS", "CSI+GSP",
   "CSI+OTS", "CSI"];

    Service {
        # Service-specific security configuration.
        ...
10      orb_plugins = ["local_log_stream", "iiop_profile",
   "giop", "iiop_tls", "ots", "gsp"];

11      plugins:Service:iiop_tls:port = "0";
        plugins:Service:iiop_tls:host = "ServiceHost";

        # Configuration of CSI and GSP plug-ins.
12      policies:csi:auth_over_transport:target_requires =
   "EstablishTrustInClient";
        policies:csi:auth_over_transport:target_supports =
   "EstablishTrustInClient";
```

Example 26: *Typical Service Configuration for the Application ORB*

```
        policies:csi:auth_over_transport:server_domain_name =
    "IONA";
        policies:csi:auth_over_transport:client_supports =
    "EstablishTrustInClient";

13      principal_sponsor:csi:use_principal_sponsor = "true";
        principal_sponsor:csi:auth_method_id = "GSSUPMech";
        principal_sponsor:csi:auth_method_data =
    ["username=IONAServiceAdmin", "password=service",
    "domain=IONA"];

14      plugins:gsp:action_role_mapping_file =
    "file:///vob/art/etc/domains/filedomain-secure-is2-tls/allow_
    all_authenticated_clients_action_role_mapping.xml";

15      plugins:gsp:authorization_realm = "IONAGlobalRealm";
        ...
    };
    ...
};
```

The preceding service configuration can be explained as follows:

1. Make sure that the `binding:client_binding_list` variable includes bindings with the `IIOP_TLS` and `CSI` interceptors. You can use the value of the `binding:client_binding_list` shown here.
2. The SSL/TLS mechanism policy specifies the default security protocol version and the available cipher suites—see "Specifying Cipher Suites" on page 343.
3. An SSL/TLS application needs a list of trusted CA certificates, which it uses to determine whether or not to trust certificates received from other SSL/TLS applications. You should edit the `policies:trusted_ca_list_policy` variable to point at a list of trusted certificate authority (CA) certificates. See "Specifying Trusted CA Certificates" on page 361.

> **Note:** If using Schannel as the underlying SSL/TLS toolkit (Windows only), the `policies:trusted_ca_list_policy` variable is ignored. Within Schannel, the trusted root CA certificates are obtained from the Windows certificate store.

4. The Orbix services all require an X.509 certificate. Hence, this line enables the SSL/TLS principal sponsor, which specifies a certificate for the application.
5. This line specifies that the X.509 certificate is contained in a PKCS#12 file. For alternative methods, see "Specifying an Application's Own Certificate" on page 363.

   Note: If using Schannel as the underlying SSL/TLS toolkit (Windows only), the `principal_sponsor:auth_method_id` value must be `security_label` instead of `pkcs12_file`.

6. Replace the X.509 certificate, by editing the `filename` option in the `principal_sponsor:auth_method_data` configuration variable to point at a custom X.509 certificate. The `filename` value should be initialized with the location of a certificate file in PKCS#12 format—see "Specifying an Application's Own Certificate" on page 363 for more details.

   Note: If using Schannel as the underlying SSL/TLS toolkit (Windows only), you would set the `label` option instead of the `filename` option in the `principal_sponsor:auth_method_data` configuration variable. The `label` specifies the common name (CN) from the application certificate's subject DN.

7. The following two lines set the *required* options and the *supported* options for the *target* secure invocation policy. In this example, which is a *secure domain*, the target policies specify that the application will accept secure connections only.

   Alternatively, in a *semi-secure domain* the target secure invocation policy would be set as follows:

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

8. The following two lines set the *required* options and the *supported* options for the *client* secure invocation policy. In this example, which is a *secure domain*, the client policies require the connection to open secure connections only.

   Alternatively, in a *semi-secure domain* the client secure invocation policy would be set as follows:

```
policies:client_secure_invocation_policy:requires =
   ["NoProtection"];
policies:client_secure_invocation_policy:supports =
   ["NoProtection", "Confidentiality",
   "EstablishTrustInTarget", "EstablishTrustInClient",
   "DetectMisordering", "DetectReplay", "Integrity"];
```

9. Make sure that the `binding:server_binding_list` variable includes bindings with the CSI and GSP interceptors. You can use the value of the `binding:server_binding_list` shown here.
10. Make sure that the `orb_plugins` variable in this configuration scope includes both the `iiop_tls` plug-in and the `gsp` 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 before the `iiop_tls` plug-in in the ORB plug-ins list.

11. The IIOP/TLS IP port is set to `0` in this example, because the node daemon is responsible for allocating the port dynamically (on demand activation). Services that are not activated on demand (for example, the locator) will be allocated a specific IP port.
12. In this example (secure domain), the CSI policies are set up in such a way that clients are required to provide a username and password to log on to the service.

    Alternatively, in a semi-secure domain the `policies:csi:auth_over_transport:target_requires` variable is set

to an empty string, "", implying that clients are *not* required to provide a username and password to the service. For example:

```
policies:csi:auth_over_transport:server_domain_name =
   "IONA";
policies:csi:auth_over_transport:target_supports =
   "EstablishTrustInClient";
policies:csi:auth_over_transport:target_requires = "";
policies:csi:auth_over_transport:client_supports =
   "EstablishTrustInClient";
```

13. The CSI principal sponsor sets a username, a password and a domain, which the server uses when acting in a client role to connect to other applications. The `principal_sponsor:csi:auth_method_data` variable is set as follows:
    - *username*—has the value `IONAServiceAdmin`. When using the default ACLs (see "Default Access Control Lists" on page 227), the `IONAServiceAdmin` user enjoys unrestricted access to all of the core Orbix services.
    - *password*—in this example, the CSI password is provided directly in the configuration file. For alternative ways of specifying the CSI password, see "Providing a Username and Password" on page 420.
    - *domain*—has the value `IONA`. The CSI authentication domain must match the target server's domain name, as specified by the `policies:csi:auth_over_transport:server_domain_name` configuration variable, or could be an empty string (acts as a wildcard).
14. The `action_role_mapping` configuration variable specifies the location of an action-role mapping that controls access to the IDL interfaces implemented by the server. The file location is specified in an URL format, for example:
    `file:///security_admin/action_role_mapping.xml` (UNIX) or `file:///c:/security_admin/action_role_mapping.xml` (Windows).
    For more details about the action-role mapping file, see "CORBA Action-Role Mapping ACL" on page 194.

15. This configuration setting specifies the iSF authorization realm, *AuthzRealm*, to which this server belongs (the default is `IONAGlobalRealm`). For more details about iSF authorization realms, see "iSF Authorization Realms" on page 177.

Configuration settings for the internal ORB

Example 27 shows the configuration settings for the internal ORB. These settings enable the management service to monitor the Orbix services. All of the settings for the internal ORB are intended to configure the server end of a connection. The internal ORB does *not* open any connections to other processes.

Example 27: *Typical Service Configuration for the Internal ORB*

```
# Orbix Configuration File
...
IT_POAInternalORB
{
    principal_sponsor:use_principal_sponsor = "true";
    principal_sponsor:auth_method_id = "pkcs12_file";
    principal_sponsor:auth_method_data =
1   "%{SERVICES_AUTH_METHOD_DATA}";

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

    policies:client_secure_invocation_policy:requires =
   ["Confidentiality", "EstablishTrustInTarget",
   "DetectMisordering", "DetectReplay", "Integrity"];
    policies:client_secure_invocation_policy:supports =
   ["Confidentiality", "EstablishTrustInClient",
   "EstablishTrustInTarget", "DetectMisordering",
   "DetectReplay", "Integrity"];

    binding:server_binding_list = ["CSI+GSP+OTS", "CSI+GSP",
   "CSI+OTS", "CSI"];

    policies:csi:auth_over_transport:target_requires =
   "EstablishTrustInClient";
```

Example 27: *Typical Service Configuration for the Internal ORB*

```
    policies:csi:auth_over_transport:target_supports =
   "EstablishTrustInClient";
    policies:csi:auth_over_transport:server_domain_name =
   "IONA";

    iona_services
    {
        Service
        {
2           orb_plugins = ["local_log_stream", "iiop_profile",
   "giop", "iiop_tls", "ots", "gsp"];

            plugins:local_log_stream:filename =
   "/vob/art/var/filedomain-secure-is2-tls/logs/IT_POAInternalOR
   Bifr.log";

3           plugins:gsp:action_role_mapping_file =
   "file:///vob/art/etc/domains/filedomain-secure-is2-tls/allow_
   all_authenticated_clients_action_role_mapping.xml";
        };
        ...
    };
};
```

The preceding internal ORB configuration can be explained as follows:

1. The internal ORB's principal sponsor should be configured with an X.509 certificate suitable for a secure Orbix service.

   Note: Instead of using the principal sponsor here, you could set the `plugins:security:share_credentials_across_orbs` configuration variable instead. See "Security Configuration" on page 485.

2. Make sure that the `orb_plugins` variable in this configuration scope includes both the `iiop_tls` plug-in and the `gsp` plug-in.
3. The internal ORB uses the `allow_all_authenticated_clients_action_role_mapping.xml` file for access control. This configuration gives unrestricted access to all authenticated clients.

Configuring the Security Service

Overview

This section describes how to configure the Orbix security service. This service is configured somewhat differently from the others. For example, because the gsp plug-in contacts the security service to perform authentication, the gsp plug-in must be *excluded* from the security service's own orb_plugins list in order to avoid a circular dependency.

Configuration settings for application ORB

Example 28 shows the configuration settings for the Orbix security service. These settings configure the application ORB—that is, these settings determine the ordinary runtime behavior of the service.

Example 28: *Security Service Configuration for the Application ORB*

```
# Orbix Configuration File
...
1  # General configuration at root scope.
...
2  initial_references:IT_SecurityService:reference = "IOR: ...";
...
iona_services {
3      # Common SSL/TLS security settings.
       ...
       security
       {
           ...
           iS2Host {
               ...
4              plugins:security:iiop_tls:port = "53112";
               plugins:security:iiop_tls:host = "iS2Host";

5              orb_plugins = ["local_log_stream", "iiop_profile",
   "giop", "iiop_tls"];

6
   policies:iiop_tls:target_secure_invocation_policy:requires =
   ["Integrity", "Confidentiality", "DetectReplay",
   "DetectMisordering", "EstablishTrustInClient"];
```

Example 28: *Security Service Configuration for the Application ORB*

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

            server
            {
7             orb_plugins = ["local_log_stream", "iiop_profile",
   "giop", "iiop_tls", "it_servlet_binding_manager",
   "it_deployer", "it_servlet_context", "it_http_sessions",
   "it_servlet_filters", "http", "https", "it_servlet_dispatch",
   "it_exception_mapping", "it_naming_context",
   "it_web_security", "it_web_app_activator",
   "it_default_servlet_binding", "it_character_encoding",
   "it_locale", "it_classloader_mapping"];
            };
        };
        ...
    };
};
```

The preceding security service configuration can be explained as follows:

1. The security service's root configuration settings are the same as in Example 26 on page 216.
2. The `IT_SecurityService` initial reference specifies the IOR that CORBA applications use to talk to the security service.
3. The common configuration settings (in the `iona_services` scope) are the same as in Example 26 on page 216.
4. The `plugins:security:iiop_tls:port` variable specifies the IP port where the security service listens for secure connections.

> **Note:** If you want to change the security service's listening port, you would also have to update the IOR in the `initial_references:IT_SecurityService:reference` setting. You could regenerate the IOR by re-running the `itconfigure` utility.

5. This `orb_plugins` setting is required here for technical reasons.

   Specifically, the Orbix security service is bootstrapped in two stages, as follows:

   i. In the first stage, the generic server (implemented in C++) instantiates an ORB with the `iona_services.security.`*`iS2Host`* configuration scope, loading a minimal set of ORB plug-ins (this `orb_plugins` setting).

   ii. In the second stage, the generic server spawns a Java process, which instantiates an ORB with the `iona_services.security.`*`iS2Host`*`.server` configuration scope, loading the full set of ORB plug-ins.

6. The IIOP/TLS target secure invocation policy requires a strong quality of protection for incoming connections.

7. 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) and the `http` plug-in (insecure HTTP) from the ORB plug-ins list. This renders the application incapable of making insecure IIOP connections and insecure HTTP connections.

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

Default Access Control Lists

Overview

When you use the `itconfigure` utility to generate a secure domain, *SecureDomain*, a collection of default action-role mapping files are generated in the `etc/domains/`*SecureDomain* directory. Each of the core Orbix services, *Service*, is associated with an action-role mapping file as follows:

- *Service*`_action_role_mapping.xml`—for a secure domain.
- *Service*`_semi_secure_action_role_mapping.xml`—for a semi-secure domain.

Two basic levels of access are defined in these ACLs: `IONAUserRole` for ordinary users; and `IONAServiceRole` for administrators.

Note: It is recommended that you check whether the default ACLs provide the level of security you need before deploying the core Orbix services in a real system.

In this section

This section contains the following subsections:

Configuration Repository ACL

Overview

The configuration repository (CFR) ACL is a special case, because it requires access control of parameter values in the IDL operations. To enable parameter-based access control, the CFR includes a special subsystem, a *request to action mapper*, which is responsible for parsing the operation parameters. In the CFR, the following kinds of parameter can be subjected to access control:

- Configuration scopes.
- Namespaces.

Note: It is recommended that you check whether the default configuration repository ACL provides the level of security you need before deploying it in a real system.

Configuration scopes

Similarly to a file domain, the CFR uses a *configuration scope* to group together related configuration settings. Configuration scopes can be nested as shown in the following example:

```
# Orbix Configuration File
demos {
    tls {
        secure_client_with_cert {
            ...
        };
    };
};
```

To reference a nested configuration scope, the period character (`.`) is used as a delimiter. For example, `demos.tls.secure_client_with_cert` refers to the innermost configuration scope of the preceding example.

Namespaces

The CFR uses *namespaces* to represent compound variable names. For example, the `principal_sponsor:csi:auth_method_id` variable name is built up as follows:

`principal_sponsor`	Namespace.
`principal_sponsor:csi`	Namespace.
`principal_sponsor:csi:auth_method_id`	Variable name.

To represent compound names composed of namespaces, the colon character (`:`) is used as a delimiter.

IT_CFR module

The `IT_CFR` module defines some of the CFR's remotely accessible interfaces and operations (the CFR also implements the IDL modules defined in `cfr_replication.idl`). The IDL for the `IT_CFR` module is available in the following file:

OrbixInstallDir`/asp/`*Version*`/idl/orbix_pdk/cfr.idl`

For example, the `itadmin` utility calls operations from the `IT_CFR` module in order to read from and update the configuration repository. Example 29 shows an overview of the interfaces defined in the `IT_CFR` module.

Example 29: *The IT_CFR Module*

```
// IDL
...
module IT_CFR {
    interface ConfigScope { ... };
    interface Namespace { ... };
    interface ConfigRepository { ... };
    interface Listener { ... };
    interface ListenerRegistration { ... };
};
```

CompoundName type

The `IT_CFR::CompoundName` type is defined as follows:

```
// IDL
module IT_CFR {
    typedef sequence<string> CompoundName;
};
```

The CompoundName type represents configuration scopes and namespaces as follows:

- *Configuration scope*—is converted into a CompoundName by recognizing the period character (.) as a delimiter. For example, the demos.tls.secure_client_with_cert scope is converted to the following sequence of strings: demos, tls, secure_client_with_cert.
- *Namespace*—is converted into a CompoundName by recognizing the colon character (:) as a delimiter. For example, the principal_sponsor:csi:auth_method_id variable name is converted to the following sequence of strings: principal_sponsor, csi, auth_method_id.

Parameter-based access control

In order to provide a meaningful level of access control for the CFR, it is necessary to control access at the level of operation parameters; operation-based access control would not be sufficient.

For example, consider the following destroy_subscope() operation from the IT_CFR module:

```
// IDL
module IT_CFR {
    interface ConfigScope
    {
        ConfigScope destroy_subscope(
            in CompoundName name
        ) raises (CFRException);
    };
};
```

Ordinary users should not have permission to destroy critical configuration scopes such as iona_services (which holds the configuration settings for the core Orbix services). But ordinary users do need full access to at least one scope, for example demos, in order to configure their own applications. *Parameter-based access control* enables you to control access based on the value of the name parameter in the preceding operation.

To control access based on the `destroy_scope()` operation's `name` parameter, you could use the following fragment in an action-role mapping file:

```
<interface>
  <name>IDL:iona.com/IT_CFR/ConfigScope:1.0</name>
  ...
  <action-role>
      <action-name>destroy_subscope</action-name>
      <parameter-control>
          <parameter name="name" value="demos.*"/>
          <role-name>IONAUserRole</role-name>
      </parameter-control>
      ...
      <role-name>IONAServiceRole</role-name>
  </action-role>
</interface>
```

This ensures that ordinary users (represented by `IONAUserRole`) can only destroy the `demos` scope and its subscopes.

ACL for configuration scope operations

Example 30, which is extracted from the default `cfr_action_role_mapping.xml` file, shows how access control is configured for the `IT_CFR::ConfigScope` interface.

Example 30: *ACL for the IT_CFR::ConfigScope Interface*

```
<interface>
  <name>IDL:iona.com/IT_CFR/ConfigScope:1.0</name>
  <action-role>
    <action-name>*get*</action-name>
    <role-name>IONAUserRole</role-name>
  </action-role>
  <action-role>
    <action-name>scope_lookup</action-name>
    <role-name>IONAUserRole</role-name>
  </action-role>
  <action-role>
    <action-name>create_subscope</action-name>
       <parameter-control>
         <parameter name="name"
value="_it_cfr_root_scope.*"/>
         <role-name>IONAUserRole</role-name>
       </parameter-control>
          <parameter-control>
```

Example 30: *ACL for the IT_CFR::ConfigScope Interface*

```
            <parameter name="name" value="demos.*"/>
            <role-name>IONAUserRole</role-name>
          </parameter-control>
          <parameter-control>
            <parameter name="name" value="multicast_demo.*"/>
            <role-name>IONAUserRole</role-name>
          </parameter-control>
          <role-name>IONAServiceRole</role-name>
      </action-role>
      <action-role>
        <action-name>destroy_subscope</action-name>
             <parameter-control>
            <parameter name="name" value="demos.*"/>
            <role-name>IONAUserRole</role-name>
          </parameter-control>
          <parameter-control>
            <parameter name="name" value="multicast_demo.*"/>
            <role-name>IONAUserRole</role-name>
          </parameter-control>
          <role-name>IONAServiceRole</role-name>
      </action-role>
      <action-role>
        <action-name>*</action-name>
        <role-name>IONAServiceRole</role-name>
      </action-role>
    </interface>
```

Locator ACL

Overview

This subsection describes which interfaces and operations are accessible through the default locator ACL. The following alternative ACL files are generated by `itconfigure` for the locator service:

- `locator_action_role_mapping.xml` (*secure* domain).
- `locator_semi_secure_action_role_mapping.xml` (*semi-secure* domain).

Note: It is recommended that you check whether the default locator ACL provides the level of security you need before deploying it in a real system.

IONAServiceRole

The `IONAServiceRole` can access all interfaces and operations in both *secure* and *semi-secure* domains.

IONAUserRole and UnauthenticatedUserRole

The `IONAUserRole` can access the locator interfaces and operations shown in Table 5 in both *secure* and *semi-secure* domains.

Unauthenticated users (represented by the special `UnauthenticatedUserRole` in the action-role mapping file) can access the interfaces and operations shown in Table 5 in *semi-secure* domains only.

Table 5: *Locator Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)
`IT_Location::Locator`	*All*	*All*
`IT_IMRAdmin::Process`	*All*	*All*
`IT_IMRAdmin::ProcessRegistry`	*All*	*All*
`IT_IMRAdmin::Process`	*All*	*All*
`IT_IMRAdmin::ORBRegistry`	*All*	*All*

Table 5: *Locator Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)
`IT_IMRAdmin::ORB`	*All*	*All*
`IT_NamedKey::NamedKeyRegistry`	*All*	*All*
`IT_POAIMRAdmin::POA`	*All*	*All*
`IT_POAIMRAdmin::POARegistry`	*All*	*All*
`IT_LocatorAdmin::ActiveORBRegistry`	*All*	*All*
`IT_LocatorAdmin::ActiveProcessRegistry`	*All*	*All*
`IT_POALocatorAdmin::ActivePOARegistry`	*All*	*All*
`IT_POAIMRAdmin::ActivePOA`	*All*	*All*
`IT_POAIMRAdmin::POAActiveORB`	*All*	*All*
`IT_POAIMRAdmin::CachedPOA`	*All*	*All*
`IT_POAIMRAdmin::POA`	*All*	*All*
`IT_POAIMRAdmin::POACache`	*All*	*All*
`IT_NodeDaemon::NodeDaemonRegistry`	*All*	*All*
`IT_NodeDaemon::NodeDaemon`	*None*	*None*
`IT_NodeDaemon::DynamicStateRegistry`	*None*	*None*
`IT_ServerLocation::ServerValidator`	*None*	*None*
`IT_ServerLocation::EndpointCache`	*None*	*None*
`IT_LocatorAdmin::ActiveProcess`	*None*	*None*

Node Daemon ACL

Overview

This subsection describes which interfaces and operations are accessible through the default node daemon ACL. The following alternative ACL files are generated by `itconfigure` for the node daemon service:

- `node_daemon_action_role_mapping.xml` (*secure* domain).
- `node_daemon_semi_secure_action_role_mapping.xml` (*semi-secure* domain).

Note: It is recommended that you check whether the default node daemon ACL provides the level of security you need before deploying it in a real system.

IONAServiceRole

The `IONAServiceRole` can access all interfaces and operations in both *secure* and *semi-secure* domains.

IONAUserRole and UnauthenticatedUserRole

The `IONAUserRole` can access the node daemon interfaces and operations shown in Table 6 in both *secure* and *semi-secure* domains.

Unauthenticated users (represented by the special `UnauthenticatedUserRole` in the action-role mapping file) can access the interfaces and operations shown in Table 6 in *semi-secure* domains only.

Table 6: *Node Daemon Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)

`IT_NodeDaemon::NodeDaemon`	`shutdown`
	`shutdown_complete`
	`register_process`
		`shutdown`
		`shutdown_complete`
		`register_process`

`IT_NodeDaemon::ORBStateRegistry`	*None*	*None*
`IT_NodeDaemon::EndpointRegistry`	*None*	*None*
`IT_NodeDaemon::ProcessRegistry`	*None*	*None*

Table 6: *Node Daemon Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)
`IT_NodeDaemon::DynamicStateRegistry`	*All*	*All*

Naming Service ACL

Overview

This subsection describes which interfaces and operations are accessible through the default naming service ACL. The following alternative ACL files are generated by `itconfigure` for the naming service:

- `naming_action_role_mapping.xml` (*secure* domain).
- `naming_semi_secure_action_role_mapping.xml` (*semi-secure* domain).

Note: It is recommended that you check whether the default naming ACL provides the level of security you need before deploying it in a real system.

IONAServiceRole

The `IONAServiceRole` can access all interfaces and operations in both *secure* and *semi-secure* domains.

IONAUserRole and UnauthenticatedUserRole

The `IONAUserRole` can access the naming service interfaces and operations shown in Table 7 in both *secure* and *semi-secure* domains.

Unauthenticated users (represented by the special `UnauthenticatedUserRole` in the action-role mapping file) can access the interfaces and operations shown in Table 7 in *semi-secure* domains only.

Table 7: *Naming Service Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)
`IT_Naming::IT_NamingContextExt`	*All*	*All*
`IT_NamingReplication::IT_MasterNamingAdmin`	`shutdown`	`shutdown`
`IT_NamingAdmin::NamingAdmin`	`shutdown`	`shutdown`
`CosNaming::NamingContextExt`	*None*	*None*
`CosNaming::BindingIterator`	*All*	*All*

Trader Service ACL

Overview

The default action-role mappings for the *trader* service are designed to protect the service by differentiating between non-intrusive operations (for example, read operations) and intrusive operations that might threaten the integrity of the service (for example, write operations).

Two different action-role mappings are provided, as follows:

- *Secure domain*—both intrusive and non-intrusive access to the trader service is restricted to authenticated applications only.
- *Semi-secure domain*—non-intrusive access to the trader service is available to both authenticated and unauthenticated applications. Intrusive access is limited to authenticated applications only.

Note: It is recommended that you check whether the default trader ACL provides the level of security you need before deploying it in a real system.

Secure domain

In a secure domain, the trader's action-role mapping file is:

`etc/`*DomainName*`/trader_action_role_mapping.xml`

Only authorized applications can add service types and service offers. This ensures that unauthorized peers will not be able to add to the repository references to malicious applications designed to mimic the behavior and appearance of expected service offers.

Applications that need to obtain references to existing service offers must also be authenticated. This prevents unauthorized client applications from looking up services they are not allowed to use.

Note: This precaution alone is not sufficient to protect server applications from unauthorized access, because querying the trader service is not the only way to obtain references to server applications. Sensitive applications must incorporate their own security mechanisms, or be protected by the security service as well.

Access to administrative operation that could endanger the integrity of the database if accessed by unauthorized parties is restricted to roles normally used by administrators (that is, `IONAServiceRole` and `IONAAdminRole`).

Semi-secure domain

In a semi-secure domain, the trader's action-role mapping file is:

`etc/`*DomainName*`/trader_semi_secure_action_role_mapping.xml`

This mapping relaxes the settings from the secure domain, so that unauthenticated users (using either secure or insecure transports) are allowed to invoke any operations that perform *read only* queries.

Only authenticated users are allowed to invoke operations that require *write access* to the Trader's database. This ensures that no malicious application will be able to export unauthorized service types or offers (for example, server applications that mimic legitimate service offers, but instead collect information passed to them by client applications).

IONAServiceRole

The `IONAServiceRole` can access all interfaces and operations in both *secure* and *semi-secure* domains.

IONAUserRole and UnauthenticatedUserRole

The `IONAUserRole` can access the trader service interfaces and operations shown in Table 8 in both *secure* and *semi-secure* domains.

Unauthenticated users (represented by the special `UnauthenticatedUserRole` in the action-role mapping file) can access the interfaces and operations shown in Table 8 in *semi-secure* domains only.

Table 8: *Trader Service Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)

`CosTradingRepos::ServiceTypeRepository`	`add_type`
	`list_types`
	`describe_type`
	`fully_describe_type`
		`list_types`
		`describe_type`
		
		`fully_describe_type`

`CosTradingDynamic::DynamicPropEval`	*All*	*All*
`IT_Trading::IT_LookupExt`	*All*	*All*
`IT_TradingAdmin::TradingAdmin`	*None*	*None*
`CosTrading::Lookup`	*All*	*All*

Table 8: *Trader Service Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)

`CosTrading::Register`	`export`
	`withdraw`
	`describe`
	`modify`
	`withdraw_using_constraint`
		None

`CosTrading::Link`	*None*	*None*
`CosTrading::Proxy`	*All*	*None*
`CosTrading::Admin`	*None*	*None*
`CosTrading::OfferIterator`	*All*	*All*
`CosTrading::OfferIdIterator`	*None*	*None*

Event Service ACL

Overview

The default action-role mappings for the *event* service are designed to protect the service by differentiating between non-intrusive operations (for example, read operations) and intrusive operations that might threaten the integrity of the service (for example, write operations).

Two different action-role mappings are provided, as follows:

- *Secure domain*—intrusive access to the event service is restricted to authenticated applications only.
- *Semi-secure domain*—intrusive access to the event service is available to both authenticated and unauthenticated applications.

Note: It is recommended that you check whether the default events ACL provides the level of security you need before deploying it in a real system.

Secure domain

In a secure domain, the event service's action-role mapping file is:

`etc/`*DomainName*`/event_action_role_mapping.xml`

Only authenticated applications can connect to the event service for the purpose of sending or receiving events. With this security scheme in place, consumers connected to the service can trust that the events they receive are legitimate (because they are known to originate from authenticated suppliers). Suppliers that send events through the event service can trust that their events reach only legitimate consumers (because consumers are also authenticated).

Semi-secure domain

In a semi-secure domain, the event service's action-role mapping file is:

`etc/`*`DomainName`*`/event_semi_secure_action_role_mapping.xml`

The security scheme for the semi-secure domain is very permissive, because all applications have full access to the service by default. The scheme could be made more secure by restricting the role of unauthenticated applications to simple listeners (by denying them the privilege of connecting suppliers to event channels).

WARNING: The semi-secure scheme should not be used if events can carry security-sensitive information, because the identity of neither the suppliers nor the consumers can be guaranteed.

IONAServiceRole

The `IONAServiceRole` can access all interfaces and operations in both *secure* and *semi-secure* domains.

IONAUserRole and UnauthenticatedUserRole

The `IONAUserRole` can access the event service interfaces and operations shown in Table 9 in both *secure* and *semi-secure* domains.

Unauthenticated users (represented by the special `UnauthenticatedUserRole` in the action-role mapping file) can access the interfaces and operations shown in Table 9 in *semi-secure* domains only.

Table 9: *Event Service Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)

`IT_EventChannelAdminInternal::EventChannelFactory`	`_get_name`
	`_get_host`
	`shutdown`
	`create_channel`
	`find_channel`
	`find_channel_by_id`
	`list_channels`
	`create_typed_channel`
	`find_typed_channel`
	`find_typed_channel_by_id`
	`list_typed_channels`
	`create`
	`find`
	`findByRef`
	`list`
	`createTyped`
	`findTyped`
	`findByTypedRef`
	`listTyped`
		`_get_name`
		`_get_host`
		`shutdown`
		`create_channel`
		`find_channel`
		`find_channel_by_id`
		`list_channels`
		`create_typed_channel`
		`find_typed_channel`
		`find_typed_channel_by_id`
		`list_typed_channels`
		`create`
		`find`
		`findByRef`
		`list`
		`createTyped`
		`findTyped`
		`findByTypedRef`
		`listTyped`

`CosEventChannelAdmin::EventChannel`	*All*	*All*
`CosTypedEventChannelAdmin::TypedEventChannel`	*All*	*All*
`CosEventChannelAdmin::SupplierAdmin`	*All*	*All*
`CosTypedEventChannelAdmin::TypedSupplierAdmin`	*All*	*All*
`CosEventChannelAdmin::ConsumerAdmin`	*All*	*All*

Table 9: *Event Service Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)
`CosTypedEventChannelAdmin::TypedConsumerAdmin`	*All*	*All*
`CosEventChannelAdmin::ProxyPushConsumer`	*All*	*All*
`CosTypedEventChannelAdmin::TypedProxyPushConsumer`	*All*	*All*
`CosEventChannelAdmin::ProxyPushSupplier`	*All*	*All*
`CosEventChannelAdmin::ProxyPullSupplier`	*All*	*All*
`CosEventChannelAdmin::ProxyPullConsumer`	*All*	*All*

Notification Service ACL

Overview

The default action-role mappings for the *notification* service are designed to protect the service by differentiating between non-intrusive operations (for example, read operations) and intrusive operations that might threaten the integrity of the service (for example, write operations).

Two different action-role mappings are provided, as follows:

- *Secure domain*—both intrusive and non-intrusive access to the notification service are restricted to authenticated applications only.
- *Semi-secure domain*—non-intrusive access to the notification service is available to both authenticated and unauthenticated applications. Intrusive access is limited to authenticated applications only.

Note: It is recommended that you check whether the default notification ACL provides the level of security you need before deploying it in a real system.

Secure domain

In a secure domain, the event service's action-role mapping file is:

`etc/`*DomainName*`/notify_action_role_mapping.xml`

Only authenticated applications can connect to the notification service for the purpose of sending or receiving notifications. With this security scheme in place, consumers connected to the service can trust that the events they receive are legitimate (because they are known to originate from authenticated suppliers). Suppliers that send events through the notification service can trust that their events reach only legitimate consumers (because consumers are also authenticated).

Authenticated applications are allowed to create and apply event filters and mapping filters, as normal.

Authenticated applications are allowed to alter the behavior of the notification service by setting *Quality of Service* properties at any level of the service. The operations that administer the notification service are also protected by access control. Hence, these adminstration operations can only be called by authenticated applications and utilities.

Semi-secure domain

In a semi-secure domain, the event service's action-role mapping file is:

`etc/`*DomainName*`/notify_semi_secure_action_role_mapping.xml`

The security scheme for the semi-secure domain forces all event suppliers to authenticate with the notification service. However any consumer, even non-authenticated consumers, can connect to the service and receive events.

Under this security model, consumers can trust the notifications they receive to be legitimate (because they are known to originate from authenticated applications only). On the other hand, suppliers do *not* know whether the events they send will reach authenticated or unauthenticated consumers.

WARNING: The semi-secure scheme should not be used if notifications can carry security-sensitive information, because suppliers have no way of knowing the identity of consumers. Also, an insecure transport might be used to carry events to the consumers.

Operations that could potentially compromise the integrity or the functionality of the notification service are restricted to authenticated applications only.

Only authenticated peers are allowed to apply filters to objects other than proxy consumers or suppliers, since filters set at any other level could potentially be used by malicious applications to prevent events from reaching they legitimate targets.

Unauthenticated consumers have the right to decide which events they want to receive: they can still apply filters to their proxy supplier. Similarly, they have *read-only* access to filters set at the channel administration level (so that they can interpret the filtration logic of the events they receive).

IONAServiceRole

The `IONAServiceRole` can access all interfaces and operations in both *secure* and *semi-secure* domains.

IONAUserRole and UnauthenticatedUserRole

The `IONAUserRole` can access the notification service interfaces and operations shown in Table 10 in both *secure* and *semi-secure* domains.

Unauthenticated users (represented by the special `UnauthenticatedUserRole` in the action-role mapping file) can access the interfaces and operations shown in Table 10 in *semi-secure* domains only.

Table 10: *Notification Service Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)
`IT_NotifyFilterInternal::Filter`	*All*	*All*
`IT_NotifyFilterInternal::MappingFilter`	*All*	*All*
`IT_NotifyFilterInternal::FilterFactory`	*All*	*All*
`IT_NotifyComm::GroupNotifyPublish`	*None*	*None*
`IT_NotifyComm::GroupPushConsumer`	*All*	*All*
`IT_NotifyComm::GroupStructuredPushConsumer`	*All*	*All*
`IT_NotifyComm::GroupSequencePushConsumer`	*All*	*All*
`IT_NotifyChannelAdmin::IT_ProxySupplier`	*All*	*All*
`IT_NotifyChannelAdmin::NotifyProxySupplier`	*All*	*All*
`IT_NotifyChannelAdmin::ProxyPushSupplier`	*All*	*All*
`IT_NotifyChannelAdmin::StructuredProxyPushSupplier`	*All*	*All*
`IT_NotifyChannelAdmin::SequenceProxyPushSupplier`	*All*	*All*
`IT_NotifyChannelAdmin::ProxyPullSupplier`	*All*	*All*

Table 10: *Notification Service Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)
`IT_NotifyChannelAdmin::StructuredProxyPullSupplier`	*All*	*All*
`IT_NotifyChannelAdmin::SequenceProxyPullSupplier`	*All*	*All*
`IT_NotifyChannelAdmin::IT_ProxyConsumer`	*All*	*All*
`IT_NotifyChannelAdmin::NotifyProxyConsumer`	*All*	*All*
`IT_NotifyChannelAdmin::ProxyPushConsumer`	*All*	*All*
`IT_NotifyChannelAdmin::StructuredProxyPushConsumer`	*All*	*All*
`IT_NotifyChannelAdmin::SequenceProxyPushConsumer`	*All*	*All*
`IT_NotifyChannelAdmin::ProxyPullConsumer`	*All*	*All*
`IT_NotifyChannelAdmin::StructuredProxyPullConsumer`	*All*	*All*
`IT_NotifyChannelAdmin::SequenceProxyPullConsumer`	*All*	*All*

Table 10: *Notification Service Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)
IT_NotifyChannelAdmin::ConsumerAdmin	get_bridge_proxy_supplier obtain_subscription_types _for_admin _get_bridge_pull_supplier s _get_bridge_push_supplier s get_proxy_supplier obtain_notification_pull_ supplier obtain_notification_push_ supplier _get_MyID _get_MyChannel _get_MyOperator _get_priority_filter _get_lifetime_filter _get_pull_suppliers _get_push_suppliers get_qos validate_qos get_filter get_all_filters obtain_push_supplier obtain_pull_supplier destroy _set_priority_filter _set_lifetime_filter set_qos subscription_change add_filter remove_filter remove_all_filters	get_bridge_proxy_supplier obtain_subscription_types _for_admin _get_bridge_pull_supplier s _get_bridge_push_supplier s get_proxy_supplier obtain_notification_pull_ supplier obtain_notification_push_ supplier _get_MyID _get_MyChannel _get_MyOperator _get_priority_filter _get_lifetime_filter _get_pull_suppliers _get_push_suppliers get_qos validate_qos get_filter get_all_filters obtain_push_supplier obtain_pull_supplier subscription_change

Table 10: *Notification Service Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)

`IT_NotifyChannelAdmin::SupplierAdmin`	`get_bridge_proxy_consumer`
	`obtain_offered_types_for_admin`
	`_get_bridge_pull_consumers`
	`_get_bridge_push_consumers`
	`_get_MyID`
	`_get_MyChannel`
	`_get_MyOperator`
	`get_qos`
	`validate_qos`
	`get_filter`
	`get_all_filters`
	`obtain_typed_notification_pull_consumer`
	`obtain_typed_notification_push_consumer`
	`get_proxy_consumer`
	`obtain_notification_pull_consumer`
	`obtain_notification_push_consumer`
	`destroy`
	`_get_pull_consumers`
	`_get_push_consumers`
	`set_qos`
	`offer_change`
	`add_filter`
	`remove_filter`
	`remove_all_filters`
	`obtain_push_consumer`
	`obtain_pull_consumer`
		`get_bridge_proxy_consumer`
		`obtain_offered_types_for_admin`
		`_get_bridge_pull_consumers`
		`_get_bridge_push_consumers`
		`_get_MyID`
		`_get_MyChannel`
		`_get_MyOperator`
		`get_qos`
		`validate_qos`
		`get_filter`
		`get_all_filters`

`IT_NotifyChannelAdmin::Manager`	*None*	*None*
`IT_NotifyChannelAdmin::GroupProxyPushSupplier`	*All*	*All*

Table 10: *Notification Service Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)
`IT_NotifyChannelAdmin::GroupStructuredProxyPushSupplier`	*All*	*All*
`IT_NotifyChannelAdmin::GroupSequenceProxyPushSupplier`	*All*	*All*

`IT_NotifyChannelAdminInternal::EventChannel`	*All*	`obtain_offered_types`
		`obtain_subscribed_types`
		`_get_event_info`
		`get_consumeradmin`
		`get_supplieradmin`
		`get_all_consumeradmins`
		`get_all_supplieradmins`
		`_get_MyFactory`
		`_get_default_consumer_admin`
		`_get_default_supplier_admin`
		`_get_default_filter_factory`
		`get_qos`
		`validate_qos`
		`get_admin`
		`for_consumers`
		`new_for_consumers_delegate`
		`new_for_consumers`

`IT_NotifyChannelAdminInternal::EventChannelFactory`	*All*	`_get_default_filter_factory`
		`find_channel`
		`find_channel_by_id`
		`list_channels`
		`_get_manager`
		`get_all_channels`
		`get_event_channel`
		`create_named_channel`
		`create_channel`

Table 10: *Notification Service Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)
`IT_NotifyChannelAdminInternal::BridgeProxyPushSupplier`	*All*	*None*
`IT_NotifyChannelAdminInternal::BridgeProxyPushConsumer`	*All*	*None*

Basic Log Service ACL

Overview

The default action-role mappings for the *basic log* service are designed to protect the service by differentiating between non-intrusive operations (for example, read operations) and intrusive operations that might threaten the integrity of the service (for example, write operations).

Two different action-role mappings are provided, as follows:

- *Secure domain*—intrusive access to the basic log service is restricted to authenticated applications only.
- *Semi-secure domain*—intrusive access to the basic log service is available to both authenticated and unauthenticated applications.

Note: It is recommended that you check whether the default basic log ACL provides the level of security you need before deploying it in a real system.

Secure domain

In a secure domain, the basic log service's action-role mapping file is:

`etc/`*DomainName*`/basic_log_action_role_mapping.xml`

Only authenticated applications can connect to the basic log service. Authenticated applications can create new logs, retrieve existing logs, or delete logs. They also have unlimited access to all of the operations related to records.

Authenticated applications also have full access to the administrative functions of the logs (for example, setting the quality of service properties on the log, changing the maximum log size, disabling a log, and so on).

Semi-secure domain

In a semi-secure domain, the basic log service's action-role mapping file is:

`etc/`*DomainName*`/basic_log_semi_secure_action_role_mapping.xml`

The security scheme for the semi-secure domain is very permissive, because all applications have full access to the service by default. The scheme could be made more secure by denying unauthenticated peers access to some of the *write* operations of the services (such as log creation or deletion).

IONAServiceRole

The `IONAServiceRole` can access all interfaces and operations in both *secure* and *semi-secure* domains.

IONAUserRole and UnauthenticatedUserRole

The `IONAUserRole` can access the basic log service interfaces and operations shown in Table 11 in both *secure* and *semi-secure* domains.

Unauthenticated users (represented by the special `UnauthenticatedUserRole` in the action-role mapping file) can access the interfaces and operations shown in Table 11 in *semi-secure* domains only.

Table 11: *Basic Log Service Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)

`IT_BasicLogAdmin::BasicLogFactory`	`_get_manager`
	`create`
	`create_with_id`
	`list_logs`
	`find_log`
	`list_logs_by_id`
		`_get_manager`
		`create`
		`create_with_id`
		`list_logs`
		`find_log`
		`list_logs_by_id`

`IT_MessagingAdmin::Manager`	`_get_name`
	`_get_host`
	`shutdown`
		`_get_name`
		`_get_host`
		`shutdown`

`DsLogAdmin::BasicLog`	*All*	*All*[a]
`DsLogAdmin::Iterator`	`get` `destroy`	`get` `destroy`

a. Security could be tightened at this level by removing access to the destroy operation, for example, or to some of the operations used to access log records (see operations inherited from the DsLogAdmin::Log interface).

Event Log Service ACL

Overview

The default action-role mappings for the *event log* service are designed to protect the service by differentiating between non-intrusive operations (for example, read operations) and intrusive operations that might threaten the integrity of the service (for example, write operations).

Two different action-role mappings are provided, as follows:

- *Secure domain*—intrusive access to the event log service is restricted to authenticated applications only.
- *Semi-secure domain*—intrusive access to the event log service is available to both authenticated and unauthenticated applications.

Note: It is recommended that you check whether the default event log ACL provides the level of security you need before deploying it in a real system.

Secure domain

In a secure domain, the event log service's action-role mapping file is:

`etc/`*DomainName*`/event_log_action_role_mapping.xml`

Only authenticated applications can connect to the event log service. With this security scheme in place, consumers connected to the built-in event channel can trust that the events they receive are legitimate (because they are known to originate from authenticated suppliers). Event suppliers can trust that their events will be sent only to legitimate consumers (because consumers are also authenticated).

Authenticated applications can create new logs, retrieve existing logs, or delete logs.

Authenticated applications also have full access to the administrative functions of the logs (for example, setting the quality of service properties on the log, changing the maximum log size, disabling a log, and so on).

Semi-secure domain

In a semi-secure domain, the event log service's action-role mapping file is:

`etc/`*DomainName*`/event_log_semi_secure_action_role_mapping.xml`

The security scheme for the semi-secure domain is very permissive, since by default all applications have full access to the service. This scheme could be made more secure by restricting the role of unauthenticated applications to simple listeners (by denying them the privilege of connecting suppliers to the event channel as well as restricting write access to the logs and log records).

The semi-secure scheme should not be used if events carry security-sensitive information, because the identity of neither the suppliers or the consumer can be guaranteed. The integrity of the logs cannot be guaranteed since unauthenticated peers have access to all of the *write* operations and can alter the content of the logs.

IONAServiceRole

The `IONAServiceRole` can access all interfaces and operations in both *secure* and *semi-secure* domains.

IONAUserRole and UnauthenticatedUserRole

The `IONAUserRole` can access the event log service interfaces and operations shown in Table 12 in both *secure* and *semi-secure* domains.

Unauthenticated users (represented by the special `UnauthenticatedUserRole` in the action-role mapping file) can access the interfaces and operations shown in Table 12 in *semi-secure* domains only.

Table 12: *Event Log Service Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)

`IT_EventLogAdmin::EventLogFactory`	`_get_manager`
	`create`
	`create_with_id`
	`list_logs`
	`find_log`
	`list_logs_by_id`
	`obtain_push_supplier`
	`obtain_pull_supplier`
		`_get_manager`
		`create`
		`create_with_id`
		`list_logs`
		`find_log`
		`list_logs_by_id`
		`obtain_push_supplier`
		`obtain_pull_supplier`

Table 12: *Event Log Service Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)

`IT_MessagingAdmin::Manager`	`_get_name`
	`_get_host`
	`shutdown`
		`_get_name`
		`_get_host`
		`shutdown`

`DsEventLogAdmin::EventLog`	*All*	*All*
`DsLogAdmin::Iterator`	`get` `destroy`	`get` `destroy`
`CosEventChannelAdmin::ConsumerAdmin`	*All*	*All*
`CosEventChannelAdmin::SupplierAdmin`	*All*	*All*
`CosEventChannelAdmin::ProxyPushSupplier`	*All*	*All*
`CosEventChannelAdmin::ProxyPullConsumer`	*All*	*All*
`CosEventChannelAdmin::ProxyPullSupplier`	*All*	*All*
`CosEventChannelAdmin::ProxyPushConsumer`	*All*	*All*

Notify Log Service ACL

Overview

The default action-role mappings for the *notify log* service are designed to protect the service by differentiating between non-intrusive operations (for example, read operations) and intrusive operations that might threaten the integrity of the service (for example, write operations).

Two different action-role mappings are provided, as follows:

- *Secure domain*—both intrusive and non-intrusive access to the notify log service are restricted to authenticated applications only.
- *Semi-secure domain*—non-intrusive access to the notify log service is available to both authenticated and unauthenticated applications. Intrusive access is limited to authenticated applications only.

Note: It is recommended that you check whether the default notify log ACL provides the level of security you need before deploying it in a real system.

Secure domain

In a secure domain, the notify log service's action-role mapping file is:

`etc/`*`DomainName`*`/notify_log_action_role_mapping.xml`

Only authenticated applications can connect to the notify log service. With this security scheme in place, consumers connected to the built-in event channel can trust that the events they receive are legitimate (because they are known to originate from authenticated suppliers). Suppliers that send events through the notification service can trust that their events will reach only legitimate consumers (because consumers are also authenticated).

Authenticated applications can create new logs, retrieve existing logs, or delete logs.

Authenticated applications also have full access to the administrative functions of the logs (for example, setting the quality of service properties on the log, changing the maximum log size, disabling a log, and so on).

Authenticated applications are allowed to create and apply both types of filters supported by the service: log filters (which decide which events get logged) and notification-style filters (which decide which kind of events pass through the built-in event channel).

Semi-secure domain

In a semi-secure domain, the notify log service's action-role mapping file is:

`etc/`*`DomainName`*`/notify_log_semi_secure_action_role_mapping.xml`

The security scheme for the semi-secure domain requires event suppliers (applications that create logs or write log records) to authenticate with the notify log service. Any consumer (even if unauthenticated) can connect to the service, however, in order to receive events and access the logs.

Only authenticated applications (normally event suppliers) can create new logs or alter the list of existing logs (for example, by removing logs). This ensures that unauthenticated applications are not able to interfere with the logging logic or alter critical information by tampering with the service's database (by removing log entries, for example).

With this semi-secure scheme, consumers are able to trust the notifications they receive from the built-in event channel to be legitimate (because the events must have originated from an authenticated application). Consumers can also trust all logs to be genuine. On the other hand, suppliers do not know whether the events they send and/or the logs they create will reach authenticated and/or unauthenticated consumers.

Unauthenticated applications have unlimited *read-only* access to all the properties of the service and the logs. They can receive events from the built-in channel, access the list of existing logs and obtain records from any existing log. Unauthenticated applications can also examine, but not change, the filtering logic applied to the service. However, even unauthenticated consumers can decide which events they want to receive by applying filters to their proxy supplier.

Note: This semi-secure scheme allows unauthenticated applications to create filters. This is a safe policy, because the unauthenticated applications cannot apply the newly created filters in places they are not supposed to.

IONAServiceRole

The `IONAServiceRole` can access all interfaces and operations in both *secure* and *semi-secure* domains.

IONAUserRole and UnauthenticatedUserRole

The `IONAUserRole` can access the notify log service interfaces and operations shown in Table 13 in both *secure* and *semi-secure* domains.

Unauthenticated users (represented by the special `UnauthenticatedUserRole` in the action-role mapping file) can access the interfaces and operations shown in Table 13 in *semi-secure* domains only.

Table 13: *Notify Log Service Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)

`IT_NotifyLogAdmin::NotifyLog`	*All*	`_non_existent`
		`obtain_offered_types`
		`obtain_subscribed_types`
		`get_filter`
		`my_factory`
		`id`
		`get_log_qos`
		`get_max_record_life`
		`get_max_size`
		`get_current_size`
		`get_n_records`
		`get_log_full_action`
		`get_administrative_state`
		`get_forwarding_state`
		`get_operational_state`
		`get_interval`
		`get_availability_status`
		`get_capacity_alarm_thresholds`
		`get_week_mask`
		`query`
		`retrieve`
		`match`
		`get_record_attribute`
		`get_consumeradmin`
		`get_supplieradmin`
		`get_all_consumeradmins`
		`get_all_supplieradmins`
		`_get_MyFactory`
		`_get_default_consumer_admin`
		`_get_default_supplier_admin`
		`_get_default_filter_factory`
		`get_qos`
		`validate_qos`
		`get_admin`
		`for_consumers`
		`new_for_consumers`

Table 13: *Notify Log Service Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)

`IT_NotifyLogAdmin::NotifyLogFactory`	`_get_default_filter_factory`
	`_get_manager`
	`create`
	`create_with_id`
	`list_logs`
	`find_log`
	`list_logs_by_id`
	`get_proxy_supplier`
	`obtain_notification_pull_supplier`
	`obtain_notification_push_supplier`
	`_get_MyID`
	`_get_MyChannel`
	`_get_MyOperator`
	`_get_priority_filter`
	`_get_lifetime_filter`
	`_get_pull_suppliers`
	`_get_push_suppliers`
	`get_qos`
	`validate_qos`
	`get_filter`
	`get_all_filters`
	`obtain_push_supplier`
	`obtain_pull_supplier`
	`destroy`
	`_set_priority_filter`
	`_set_lifetime_filter`
	`set_qos`
	`subscription_change`
	`add_filter`
	`remove_filter`
	`remove_all_filters`
		`_get_default_filter_factory`
		`_get_manager`
		
		`list_logs`
		`find_log`
		`list_logs_by_id`
		`get_proxy_supplier`
		`obtain_notification_pull_supplier`
		`obtain_notification_push_supplier`
		`_get_MyID`
		`_get_MyChannel`
		`_get_MyOperator`
		`_get_priority_filter`
		`_get_lifetime_filter`
		`_get_pull_suppliers`
		`_get_push_suppliers`
		`get_qos`
		`validate_qos`
		`get_filter`
		`get_all_filters`
		`obtain_push_supplier`
		`obtain_pull_supplier`
		
		`subscription_change`

`IT_MessagingAdmin::Manager`	*All*	*None*
`DsLogAdmin::Iterator`	`get` `destroy`	`get` `destroy`

Table 13: *Notify Log Service Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)

`IT_NotifyChannelAdmin::ConsumerAdmin`	*All*	`get_bridge_proxy_supplier`
		`obtain_subscription_types _for_admin`
		`_get_bridge_pull_supplier s`
		`_get_bridge_push_supplier s`
		`get_proxy_supplier`
		`obtain_notification_pull_ supplier`
		`obtain_notification_push_ supplier`
		`_get_MyID`
		`_get_MyChannel`
		`_get_MyOperator`
		`_get_priority_filter`
		`_get_lifetime_filter`
		`_get_pull_suppliers`
		`_get_push_suppliers`
		`get_qos`
		`validate_qos`
		`get_filter`
		`get_all_filters`
		`obtain_push_supplier`
		`obtain_pull_supplier`
		`subscription_change`

Table 13: *Notify Log Service Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)

`IT_NotifyChannelAdmin::SupplierAdmin`	*All*	`get_bridge_proxy_consumer`
		`obtain_offered_types_for_admin`
		`_get_bridge_pull_consumers`
		`_get_bridge_push_consumers`
		`_get_MyID`
		`_get_MyChannel`
		`_get_MyOperator`
		`get_qos`
		`validate_qos`
		`get_filter`
		`get_all_filters`

`IT_NotifyChannelAdmin::ProxyPushSupplier`	*All*	*All*
`IT_NotifyChannelAdmin::StructuredProxyPushSupplier`	*All*	*All*
`IT_NotifyChannelAdmin::SequenceProxyPushSupplier`	*All*	*All*
`IT_NotifyChannelAdmin::ProxyPullSupplier`	*All*	*All*
`IT_NotifyChannelAdmin::StructuredProxyPullSupplier`	*All*	*All*
`IT_NotifyChannelAdmin::SequenceProxyPullSupplier`	*All*	*All*
`IT_NotifyChannelAdmin::ProxyPushConsumer`	*All*	*All*
`IT_NotifyChannelAdmin::StructuredProxyPushConsumer`	*All*	*All*

Table 13: *Notify Log Service Interfaces and Operations Accessible to the IONAUserRole and the UnauthenticatedUserRole*

IDL Interface	IONAUserRole Accessible Operations (Secure and semi-secure)	UnauthenticatedUserRole Accessible Operations (Semi-secure only)
`IT_NotifyChannelAdmin::SequenceProxyPushConsumer`	*All*	*All*
`IT_NotifyChannelAdmin::ProxyPullConsumer`	*All*	*All*
`IT_NotifyChannelAdmin::StructuredProxyPullConsumer`	*All*	*All*
`IT_NotifyChannelAdmin::SequenceProxyPullConsumer`	*All*	*All*
`IT_NotifyChannelAdmin::GroupProxyPushSupplier`	*All*	*All*
`IT_NotifyChannelAdmin::GroupStructuredProxyPushSupplier`	*All*	*All*
`IT_NotifyChannelAdmin::GroupSequenceProxyPushSupplier`	*All*	*All*
`IT_NotifyFilterInternal::Filter`	*All*	*All*
`IT_NotifyFilterInternal::MappingFilter`	*All*	*All*
`IT_NotifyFilterInternal::FilterFactory`	*All*	*All*

Part III

SSL/TLS Administration

In this part

This part contains the following chapters:

CHAPTER 10

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:

- "Baltimore Toolkit for C++ and Java" on page 271.
- "Schannel Toolkit for C++" on page 272.

JSSE/JCE architecture for Java applications

To replace the SSL/TLS toolkit underlying your Orbix Java applications, you can configure Orbix to use the JSSE toolkit option. For details, see:

- "JSSE/JCE Architecture" on page 274.

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

Baltimore Toolkit for C++ and Java

Overview

This section describes how to configure Orbix to use the SSL/TLS toolkit from Baltimore technologies.

Default SSL/TLS toolkit

Orbix applications use the Baltimore SSL/TLS toolkit by default. Hence, there is no need to alter your Orbix configuration to use this toolkit.

Choosing the Baltimore toolkit for C++ applications

To ensure that Orbix uses the Baltimore toolkit for C++ applications, you can optionally add the settings shown in Example 31 to your Orbix configuration. These settings are *not* necessary, however, because the Baltimore toolkit is used by default.

Example 31: *Configuring Orbix to use the Baltimore Toolkit in C++*

```
# Orbix configuration file
initial_references:IT_TLS_Toolkit:plugin = "baltimore_toolkit";
plugins:baltimore_toolkit:shlib_name = "it_tls_baltimore";
```

Choosing the Baltimore toolkit for Java applications

To ensure that Orbix uses the Baltimore toolkit for Java applications, you can optionally add the setting shown in Example 32 to your Orbix configuration. This setting is *not* necessary, however, because the Baltimore toolkit is used by default.

Example 32: *Configuring Orbix to use the Baltimore Toolkit in Java*

```
# Orbix configuration file
plugins:atli2_tls:use_jsse_tk = "false";
```

References

You can find out more about Baltimore Technologies' security products from their Web site: http://www.baltimore.com/.

Schannel Toolkit for C++

Overview

This section describes how to configure Orbix to use the Schannel toolkit from Microsoft. Schannel is a software implementation of the SSL/TLS security protocol which uses the Microsoft Crypto API (MS CAPI) to implement the cryptographic functionality required by SSL/TLS.

Note: The Schannel toolkit is available only on Windows platforms for the purpose of securing C++ applications.

The following special features are available to C++ applications that use the Schannel toolkit:

- Smart cards.
- Schannel certificate stores.

Smart cards

Because almost all smart card hardware vendors make their devices available as an MS CAPI Cryptographic Service Provider (CSP), applications that use Schannel can access a very wide range of cyptographic devices and smart cards.

Schannel certificate stores

With Schannel, application certificates and trusted CA certificates are stored in the standard Windows certificate store, thus simplifying the administration of certificates on Windows platforms.

Choosing the Schannel toolkit

You can specify that Orbix uses the Schannel toolkit by adding the settings shown in Example 31 to your Orbix configuration.

Example 33: *Configuring Orbix to use the Schannel Toolkit*

```
# Orbix configuration file
initial_references:IT_TLS_Toolkit:plugin = "schannel_toolkit";
plugins:schannel_toolkit:shlib_name = "it_tls_schannel";
```

Administration impact of switching to Schannel

Orbix toolkit replaceability is designed to be as transparent as possible to the user. Nevertheless, there are some aspects of administration that are affected by the switch to using Schannel, as follows:

- "Deploying Trusted Certificate Authorities" on page 320.
- "Deploying Application Certificates" on page 321.
- "Deploying Certificates in Smart Cards" on page 324.
- "Providing a Pass Phrase or PIN" on page 367.

Programming impact of switching to Schannel

The following aspects of security programming are affected by the switch to using Schannel:

- "Creating SSL/TLS Credentials" on page 466.

JSSE/JCE Architecture

Overview

The Java Cryptography Extension (JCE) is a pluggable framework that allows you to replace the Java security implementation with arbitrary third-party toolkits, known as *security providers*.

By default, Orbix does *not* use the JSSE/JCE framework (it accesses the Baltimore toolkit directly instead). It is possible, however, to configure Orbix to use the JSSE/JCE architecture, as described in this section.

Prerequisites

The following prerequisites must be satisfied to use the JSSE/JCE architecture with Orbix:

1. *Install J2SE (JDK) 1.4.x*—the JSSE API used internally by Orbix has changed between J2SE 1.3 and 1.4. To support the existing Orbix TLS functionality, it is necessary to use the newer JSSE/TLS API from J2SE 1.4. Security providers must support this new API in order to be compatible with Orbix.

   Note: Security providers that implement custom APIs might not work with Orbix.

2. *Install the* unlimited strength *JCE policy files*—these files allow you to use security providers that implement strong cryptography. See the following reference:

   http://java.sun.com/products/jce/#UnlimitedDownload

Using JSSE/JCE with Orbix

To use the JSSE/JCE architecture with your Orbix Java applications and to install a third-party security provider, perform the following steps:

Step	Action
1	Configure Orbix to use JSSE/JCE.
2	Configure the java.security file.
3	Install the provider JAR files.

Configure Orbix to use JSSE/JCE

To configure Orbix to use JSSE/JCE, add the setting shown in Example 34 to your Orbix configuration.

Example 34: *Configuring Orbix to use JSSE/JCE*

```
# Orbix configuration file
plugins:atli2_tls:use_jsse_tk = "true";
```

Configure the java.security file

JCE security providers are selected by specifying a list of security provider classes in the `java.security` file, which is found at the following location:

JAVA_HOME`/lib/security/java.security`

For example, to use the Sun JSSE security implementation you would configure `java.security` as shown in Example 35.

Example 35: *Sample Java Security File*

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.sun.net.ssl.internal.ssl.Provider
security.provider.3=com.sun.rsajca.Provider
security.provider.4=com.sun.crypto.provider.SunJCE
security.provider.5=sun.security.jgss.SunProvider
```

The properties in Example 35 are organized as a prioritized list. When JCE looks for the implementation of a Java security interface, it first checks the class specified by `security.provider.1` and then proceeds to the higher positions until it finds an interface implementation. Hence, it is possible for different aspects of security to be implemented by different security providers.

For more details, see Configuring the Provider (http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html#Configuring).

Install the provider JAR files

Generally, you need to add the third-party JAR files to your CLASSPATH to make a security provider accessible to Orbix. Please follow the installation instructions provided by your third-party security provider.

For more details about installing the provider classes, see:

http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html#InstallProv

Add a provider by programming

The JCE architecture provides an API that enables you to add a security provider by programming—see Configuring the Provider (http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html#InstallProv). The `java.security.Security` API can be used instead of or in addition to configuring the `java.security` file.

java.security.Security.addProvider()

Add a security provider to the next available position.

java.security.Security.insertProviderAt()

Add a security provider to the specified position. The succeeding security providers are shifted down by one position.

For more details, see the `java.security.Security` reference page:

http://java.sun.com/j2se/1.4.2/docs/api/java/security/Security.html

Using a third-party certificate store

By default, Orbix continues to use its built-in certificate storage mechanism (that is, using the principal sponsor to specify certificates) even if you install a third-party security provider that is capable of storing certificates.

You can get an Orbix application to use a third-party certificate store by configuring it as follows:

1. A basic prerequisite is to configure the third-party security provider as described in "Using JSSE/JCE with Orbix" on page 274.
2. Configure Orbix to check for a third-party certificate store by adding the following setting to your application's configuration:

```
# Orbix Configuration File
policies:tls:use_external_cert_store = "true";
```

3. Specify the name of the security provider and the name of the security protocol to use with the certificates. For example, to select the Sun JSSE security provider and the SSLv3 protocol, add the following lines to your application's configuration:

```
# Orbix Configuration File
plugins:atli2_tls:cert_store_provider = "SunJSSE";
plugins:atli2_tls:cert_store_protocol = "SSLv3";
```

Internally, Orbix passes the provider and protocol values as arguments to the `javax.net.ssl.SSLContext.getInstance()` method. *To find out the appropriate values for these settings, consult your third-party security provider documentation.*

4. Specify the key manager factory algorithm and the trust manager factory algorithm. There are two alternative approaches to setting these properties, as follows:

   - `java.security` properties file—the security provider uses these settings by default. For example, if you are using the Sun JSSE security provider, you can add the following settings to the `java.security` properties file:

   ```
   ssl.KeyManagerFactory.algorithm=SunX509
   ssl.TrustManagerFactory.algorithm=SunX509
   ```

   - Orbix configuration file—Orbix uses these settings to override the default values appearing in the `java.security` file. You would only use these configuration variables, if your `java.security` file is shared by multiple applications that need different settings.

     For example, if you are using the Sun JSSE security provider, you can add the following settings to the Orbix configuration file:

   ```
   plugins:atli2_tls:kmf_algorithm = "SunX509";
   plugins:atli2_tls:tmf_algorithm = "SunX509";
   ```

   Internally, Orbix passes the key manager factory algorithm to the `javax.net.ssl.KeyManagerFactory.getInstance()` method and the trust manager factory algorithm to the `javax.net.ssl.TrustManagerFactory.getInstance()` method. *To find out the appropriate values for these settings, consult your third-party security provider documentation.*

Logging

When using the JSSE/JCE architecture with Orbix, the log records which security provider performs an action. This is a useful debugging aid when multiple security providers are installed.

For example, the following is a log extract for an application that uses the Bouncy Castle security provider to read PKCS#12 files (`PKCS12 BC`) and the IAIK security provider to read PKCS#11 smart card credentials (`PKCS11 IAIK PKCS#11:1`).

```
11:24:15 2/20/2003
   [_it_orb_id_1@yogibear.dublin.emea.iona.com/10.2.3.6]
   (IT_ATLI_TLS:250) I - "Using the following provider: PKCS12
   BC"
11:24:21 2/20/2003
   [_it_orb_id_1@yogibear.dublin.emea.iona.com/10.2.3.6]
   (IT_TLS:201) I - Authentication succeeded using the
   IT_TLS_AUTH_METH_PKCS12_FILE method

11:24:15 2/20/2003 [_it_orb_id_1@yogibear/10.2.3.58]
   (IT_ATLI_TLS:250) I - "Using the following provider: PKCS11
   IAIK PKCS#11:1"
11:24:15 2/20/2003 [_it_orb_id_1@yogibear/10.2.3.58]
   (IT_TLS:201) I - Authentication succeeded using the
   IT_TLS_AUTH_METH_PKCS11 method
```

Troubleshooting

At the time of writing, the JSSE/JCE architecture is a relatively new technology and some of the third-party security providers have specific limitations or bugs. One approach to working around these problems is by using a combination of security providers, with different security providers implementing different aspects of security.

For example, the following general security features could be implemented by distinct security providers:

- PKCS#12 functionality—loading credentials from PKCS#12 files.
- PKCS#11 functionality—loading credentials from a smart card.
- SSL/TLS encryption.

References

For more information about Sun's JSSE/JCE architecture, see the following links:

- Java Cryptography Extension (http://java.sun.com/products/jce/index-14.html).
- J2SE (JDK) 1.4.2 Security (http://java.sun.com/j2se/1.4.2/docs/guide/security/).

- JCE Reference Guide (http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html).
- How to implement a security provider (http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/HowToImplAJCEProvider.html).
- Installing JCE providers (http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html#InstallProvider).

CHAPTER 11

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 "ASN.1 and Distinguished Names" on page 629 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.

OpenSSL software package

One software package that allows you to set up a private CA is OpenSSL, `http://www.openssl.org`. OpenSSL is derived from SSLeay, an implementation of SSL developed by Eric Young (eay@cryptsoft.com). Complete license information can be found in Appendix H on page 557. The OpenSSL package includes basic command line utilities for generating and signing certificates and these utilities are available with every installation of Orbix. Complete documentation for the OpenSSL command line utilities is available from `http://www.openssl.org/docs`.

Setting up a private CA using OpenSSL

For instructions on how to set up a private CA, see "Creating Your Own Certificates" on page 292.

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 41 shows an example of a simple certificate chain.

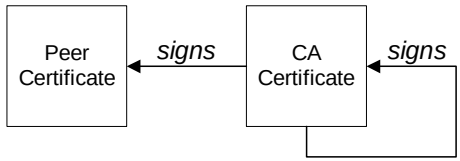

Figure 41: *A Certificate Chain of Depth 2*

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 42 shows what this certificate chain looks like.

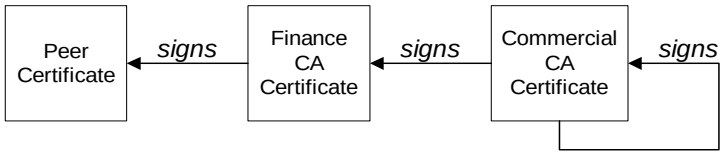

Figure 42: *A Certificate Chain of Depth 3*

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.

See "Providing a List of Trusted Certificate Authorities" on page 301.

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.

Creating a PKCS#12 file

To create a PKCS#12 file, see "Use the CA to Create Signed Certificates" on page 296.

Viewing a PKCS#12 file

To view a PKCS#12 file, *CertName*`.p12`:

```
openssl pkcs12 -in CertName.p12
```

Importing and exporting PKCS#12 files

The generated PKCS#12 files can be imported into browsers such as IE or Netscape. Exported PKCS#12 files from these browsers can be used in Orbix.

Note: Use OpenSSL v0.9.2 or later; Internet Explorer 5.0 or later; Netscape 4.7 or later.

Using the Demonstration Certificates

Location of the demonstration certificates

The Orbix certificates directory contains a set of demonstration certificates that enable you to run the Orbix example applications. The certificates are contained in this directory:

ASPInstallDir`/asp/6.0/etc/tls/x509/certs`

Default CA certificate

The CA used to sign the demonstration certificates is the default Orbix CA:

- The CA certificate is `x509/certs/ca/abigbank_ca.pem`.
- The list of trusted CA's is contained in `x509/certs/trusted_ca_lists/ca_list1.pem`. This initially contains only the `abigbank_ca.pem` CA, but other CAs can be appended.

Note: No whitespace or text is allowed in this file outside the `BEGIN/END` statements.

Certificates for demonstration programs

The PKCS#12 certificates in Table 14 are used by the Orbix demonstration programs. These certificates are located in the `x509/certs/demos` directory and signed by the `x509/certs/ca/abigbank_ca.pem` CA certificate.

Table 14: *Demonstration Certificates and Passwords*

Demonstration Certificate	Password
`certs/demos/admin.p12`	`adminpass`
`certs/demos/alice.p12`	`alicepass`
`certs/demos/bankserver.p12`	`bankserverpass`
`certs/demos/bob.p12`	`bobpass`
`certs/demos/`*CertName*`.p12`	*CertName*`pass`

Untrusted demonstration certificate

In the demonstration programs, the following certificate, `bad_guy.p12`, is used to represent a certificate from an untrusted CA:

`certs/demos/bad_guy.p12`

[REVISIT - What is the password for bad_guy.p12? I don't think it is bad_guypass.]

Certificates for the Orbix services

The Orbix services all use the same certificate, as shown in Table 15.

Table 15: *Demonstration Certificate for the Orbix Services*

Services Demonstration Certificate	Password
`certs/services/administrator.p12`	`administratorpass`

Creating Your Own Certificates

Overview

This section describes the steps involved in setting up a CA and signing certificates.

OpenSSL utilities

The steps described in this section are based on the OpenSSL command-line utilities from the OpenSSL project, `http://www.openssl.org`—see Appendix F on page 537. Further documentation of the OpenSSL command-line utilities can be obtained from `http://www.openssl.org/docs`.

Sample CA directory structure

For the purposes of illustration, the CA database is assumed to have the following directory structure:

X509CA`/ca`

X509CA`/certs`

X509CA`/newcerts`

X509CA`/crl`

Where *X509CA* is the parent directory of the CA database.

In this section

This section contains the following subsections:

Set Up Your Own CA

Substeps to perform

This section describes how to set up your own private CA. Before setting up a CA for a real deployment, read the additional notes in "Choosing a host for a private certification authority" on page 286.

To set up your own CA, perform the following substeps:

- Step 1—Add the bin directory to your PATH
- Step 2—Create the CA directory hierarchy
- Step 3—Copy and edit the openssl.cnf file
- Step 4—Initialize the CA database
- Step 5—Create a self-signed CA certificate and private key

Step 1—Add the bin directory to your PATH

On the secure CA host, add the Orbix `bin` directory to your path:

Windows

```
> set PATH=ASPInstallDir\asp\6.0\bin;%PATH%
```

UNIX

```
% PATH=ASPInstallDir/asp/6.0/bin:$PATH; export PATH
```

This step makes the `openssl` utility available from the command line.

Step 2—Create the CA directory hierarchy

Create a new directory, *X509CA*, to hold the new CA. This directory will be used to hold all of the files associated with the CA. Under the *X509CA* directory, create the following hierarchy of directories:

X509CA`/ca`

X509CA`/certs`

X509CA`/newcerts`

X509CA`/crl`

Step 3—Copy and edit the openssl.cnf file

Copy the `openssl.cnf` file to the *X509CA* directory, as follows:

Windows

```
copy ASPInstallDir\asp\6.0\etc\tls\x509\openssl.cnf
   X509CA\openssl.cnf
```

UNIX

cp *ASPInstallDir*/asp/6.0/etc/tls/x509/openssl.cnf
X509CA/openssl.cnf

Edit the openssl.cnf to reflect the directory structure of the *X509CA* directory and to identify the files used by the new CA.

Edit the [CA_default] section of the openssl.cnf file to make it look like the following:

```
##################################################################
[ CA_default ]

dir        = X509CA            # Where CA files are kept
certs      = $dir/certs  # Where issued certs are kept
crl_dir    = $dir/crl          # Where the issued crl are kept
database   = $dir/index.txt    # Database index file
new_certs_dir = $dir/newcerts  # Default place for new certs

certificate   = $dir/ca/new_ca.pem # The CA certificate
serial        = $dir/serial        # The current serial number
crl           = $dir/crl.pem       # The current CRL
private_key   = $dir/ca/new_ca_pk.pem  # The private key
RANDFILE      = $dir/ca/.rand      # Private random number file

x509_extensions = usr_cert  # The extensions to add to the cert
...
```

You might like to edit other details of the OpenSSL configuration at this point—for more details, see "The OpenSSL Configuration File" on page 547.

Step 4—Initialize the CA database

In the *X509CA* directory, initialize two files, serial and index.txt.

Windows

```
> echo 01 > serial
```

To create an empty file, index.txt, in Windows start a Windows Notepad at the command line in the *X509CA* directory, as follows:

```
> notepad index.txt
```

In response to the dialog box with the text, Cannot find the text.txt file. Do you want to create a new file?, click Yes, and close Notepad.

UNIX

```
% echo "01" > serial
% touch index.txt
```

These files are used by the CA to maintain its database of certificate files.

Note: The `index.txt` file must initially be completely empty, not even containing white space.

Step 5—Create a self-signed CA certificate and private key

Create a new self-signed CA certificate and private key:

```
openssl req -x509 -new -config
   X509CA/openssl.cnf -days 365 -out X509CA/ca/new_ca.pem
   -keyout X509CA/ca/new_ca_pk.pem
```

The command prompts you for a pass phrase for the CA private key and details of the CA distinguished name:

```
Using configuration from X509CA/openssl.cnf
Generating a 512 bit RSA private key
....+++++
.+++++
writing new private key to 'new_ca_pk.pem'
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be
incorporated into your certificate request.
What you are about to enter is what is called a Distinguished
Name or a DN. There are quite a few fields but you can leave
some blank. For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) []:IE
State or Province Name (full name) []:Co. Dublin
Locality Name (eg, city) []:Dublin
Organization Name (eg, company) []:IONA Technologies PLC
Organizational Unit Name (eg, section) []:Finance
Common Name (eg, YOUR name) []:Gordon Brown
Email Address []:gbrown@iona.com
```

Note: The security of the CA depends on the security of the private key file and private key pass phrase used in this step.

You should ensure that the file names and location of the CA certificate and private key, `new_ca.pem` and `new_ca_pk.pem`, are the same as the values specified in `openssl.cnf` (see the preceding step).

You are now ready to sign certificates with your CA.

Use the CA to Create Signed Certificates

Substeps to perform

If you have set up a private CA, as described in "Set Up Your Own CA" on page 293, you are now ready to create and sign your own certificates.

To create and sign a certificate in PKCS#12 format, *CertName*`.p12`, perform the following substeps:

- Step 1—Add the bin directory to your PATH
- Step 2—Create a certificate signing request
- Step 3—Sign the CSR
- Step 4—Concatenate the files
- Step 5—Create a PKCS#12 file
- Step 6—Repeat steps as required

Step 1—Add the bin directory to your PATH

If you have not already done so, add the Orbix `bin` directory to your path:

Windows

```
> set PATH=ASPInstallDir\asp\6.0\bin;%PATH%
```

UNIX

```
% PATH=ASPInstallDir/asp/6.0/bin:$PATH; export PATH
```

This step makes the `openssl` utility available from the command line.

Step 2—Create a certificate signing request

Create a new certificate signing request (CSR) for the *CertName*`.p12` certificate:

```
openssl req -new -config X509CA/openssl.cnf
   -days 365 -out X509CA/certs/CertName_csr.pem -keyout
   X509CA/certs/CertName_pk.pem
```

This command prompts you for a pass phrase for the certificate's private key and information about the certificate's distinguished name.

Some of the entries in the CSR distinguished name must match the values in the CA certificate (specified in the CA Policy section of the `openssl.cnf` file). The default `openssl.cnf` file requires the following entries to match:

- Country Name
- State or Province Name
- Organization Name

The Common Name must be distinct for every certificate generated by OpenSSL.

```
Using configuration from X509CA/openssl.cnf
Generating a 512 bit RSA private key
.+++++
.+++++
writing new private key to 'X509CA/certs/CertName_pk.pem'
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be
incorporated into your certificate request.
What you are about to enter is what is called a Distinguished
Name or a DN. There are quite a few fields but you can leave
some blank. For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) []:IE
State or Province Name (full name) []:Co. Dublin
Locality Name (eg, city) []:Dublin
Organization Name (eg, company) []:IONA Technologies PLC
Organizational Unit Name (eg, section) []:Systems
Common Name (eg, YOUR name) []:Orbix
Email Address []:info@iona.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:password
An optional company name []:IONA
```

Step 3—Sign the CSR

Sign the CSR using your CA:

```
openssl ca -config X509CA/openssl.cnf -days 365 -in
   X509CA/certs/CertName_csr.pem -out
   X509CA/certs/CertName.pem
```

This command requires the pass phrase for the private key associated with the `new_ca.pem` CA certificate:

```
Using configuration from X509CA/openssl.cnf
Enter PEM pass phrase:
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows
countryName           :PRINTABLE:'IE'
stateOrProvinceName   :PRINTABLE:'Co. Dublin'
localityName          :PRINTABLE:'Dublin'
```

```
organizationName      :PRINTABLE:'IONA Technologies PLC'
organizationalUnitName:PRINTABLE:'Systems'
commonName            :PRINTABLE:'Bank Server Certificate'
emailAddress          :IA5STRING:'info@iona.com'
Certificate is to be certified until May 24 13:06:57 2000 GMT (365
   days)
Sign the certificate? [y/n]:y
1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

To sign the certificate successfully, you must enter the CA private key pass phrase—see "Set Up Your Own CA" on page 293.

Step 4—Concatenate the files

Concatenate the CA certificate file, *CertName* certificate file, and *CertName*_pk.pem private key file as follows:

Windows

```
copy X509CA\ca\new_ca.pem +
   X509CA\certs\CertName.pem +
   X509CA\certs\CertName_pk.pem
   X509CA\certs\CertName_list.pem
```

UNIX

```
cat X509CA/ca/new_ca.pem
   X509CA/certs/CertName.pem
   X509CA/certs/CertName_pk.pem >
   X509CA/certs/CertName_list.pem
```

Step 5—Create a PKCS#12 file

Create a PKCS#12 file from the *CertName*_list.pem file as follows:

```
openssl pkcs12 -export -in X509CA/certs/CertName_list.pem -out
   X509CA/certs/CertName.p12 -name "New cert"
```

Step 6—Repeat steps as required

Repeat steps 2 to 5, creating a complete set of certificates for your system. A minimum set of Orbix certificates must include a set of certificates for the secure Orbix services.

Deploying Certificates

Overview

This section provides an overview of deploying X.509 certificates in a typical secure Orbix system, with detailed instructions on how to deploy certificates for different parts of the Orbix system.

In this section

This section contains the following subsections:

Overview of Certificate Deployment

Overview

Figure 43 provides an overview of the certificates used in a typical deployment of Orbix.

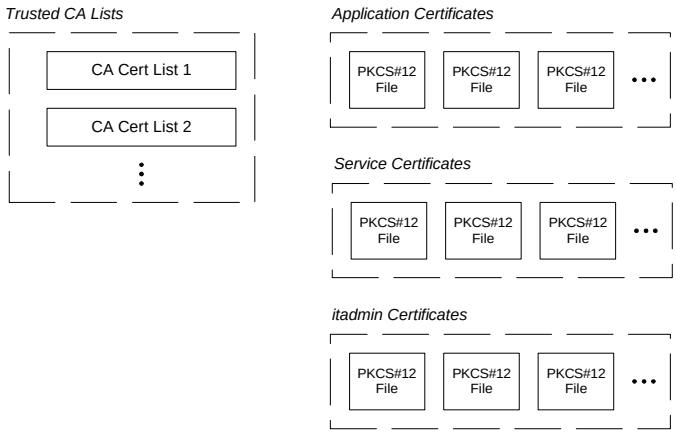

Figure 43: *Overview of Certificates in a Typical Deployed System*

Sample deployment directory structure

For the purposes of illustration, the examples in this section deploy certificates into the following sample directory structure:

X509Deploy`/trusted_ca_lists`

X509Deploy`/certs/applications`

X509Deploy`/certs/services`

X509Deploy`/certs/admin`

Where *X509Deploy* is the parent directory for the deployed certificates.

Providing a List of Trusted Certificate Authorities

Configuration variable

You can specify the list of root trusted certificates authorities by setting the `policies:iiop_tls:trusted_ca_list_policy` and `policies:https:trusted_ca_list_policy` configuration variables for IIOP/TLS and HTTPS respectively.

This variable contains a list of strings, each of which provides the filename and path of a file containing one or more trusted CA certificates. For example:

```
policies:iiop_tls:trusted_ca_list_policy =
   ["ASPInstallDir/asp/6.0/etc/tls/x509/certs/trusted_ca_lists/ca_
   list1.pem"];
```

The directory containing the trusted CA certificate lists (for example, *ASPInstallDir*`/asp/6.0/etc/tls/x509/certs/trusted_ca_lists/`) should be a secure directory.

Note: If an application supports authentication of a peer, that is a client supports `EstablishTrustInTarget`, then a file containing trusted CA certificates must be provided. If not, a `NO_RESOURCES` exception is raised.

Choosing a configuration domain

Before deploying the CA certificate on a target host, you must have access to a secure configuration domain or you can create a new domain—see the *Administrator's Guide*.

For example, if you create a secure file-based configuration domain, *SecureDomain*, you could view or modify the configuration by editing the corresponding *ASPInstallDir*`/etc/domains/`*SecureDomain*`.cfg` file.

Choosing a deployment directory

CA certificates are deployed as concatenated lists. These CA list files can be stored in any location; however, it is convenient to store them under a common deployment directory, for example:

X509Deploy`/trusted_ca_lists`

Deploying

To deploy a trusted CA certificate, perform the following steps:

Step	Action

1	If you have access to an existing secure domain, *SecureDomain*, you can append the CA certificate contents to one of the files specified in the `policies:iiop_tls:trusted_ca_list_policy` configuration variable for IIOP/TLS or in the `policies:https:trusted_ca_list_policy` configuration variable for HTTPS.
	For example, consider how to configure the IIOP/TLS protocol. If `policies:iiop_tls:trusted_ca_list_policy` lists the file, *X509Deploy*`/trusted_ca_lists/ca_list1.pem`, you can add your new CA to the `ca_list1.pem` file as follows:
	Windows
	`copy` *X509Deploy*`\trusted_ca_lists\ca_list1.pem +`
	X509CA`\ca\new_ca.pem`
	X509Deploy`\trusted_ca_lists\ca_list1.pem`
	UNIX
	`cat` *X509CA*`/ca/new_ca.pem >>`
	X509Deploy`/trusted_ca_lists/ca_list1.pem`
	The CA certificate is now deployed; hence you can skip steps 2 and 3.

2	Alternatively, you can create a new CA list file to hold your CA certificate. Copy the `new_ca.pem` certificate to the *X509Deploy*`/trusted_ca_lists` directory. Rename `new_ca.pem` to `ca_list.pem`, to remind you that this file is actually a list of certificates that happens to contain one certificate. Do *not* copy the CA private key to the target host.

3	Add the `ca_list.pem` file to your list of trusted CA files. For example, in the case of IIOP/TLS:
	`policies:iiop_tls:trusted_ca_list_policy =`
	`["`*X509Deploy*`/trusted_ca_lists/existing_list.pem",`
	`"`*X509Deploy*`/trusted_ca_lists/ca_list.pem"];`

Deploying Application Certificates

Choosing a deployment directory

Application certificates are stored as PKCS#12 files (with `.p12` suffix). The certificates can be stored in arbitrary locations; however, it is usually convenient to store the application certificates under a common deployment directory, for example:

X509Deploy`/certs/applications`

Deploying

To deploy an application certificate, *CertName*`.p12`, for an application that uses the *SampleApp* ORB name in the *DomainName* domain, perform the following steps:

Step	Action
1	Copy the application certificate, *CertName*`.p12`, to the certificates directory—for example, *X509Deploy*`/certs/applications`—on the deployment host. The certificates directory should be a secure directory that is accessible only to administrators and other privileged users.

2	Edit the *DomainName* configuration file (usually *ASPInstallDir*`/etc/domains/`*DomainName*`.cfg`). In the *SampleApp* scope, change the principal sponsor configuration to specify the *CertName*`.p12` certificate, as follows:
	`# Orbix Configuration File`
	SampleApp `{`
	`  ...`
	`  principal_sponsor:use_principal_sponsor = "true";`
	`  principal_sponsor:auth_method_id = "pkcs12_file";`
	`  principal_sponsor:auth_method_data =`
	`   ["filename=`*X509Deploy*`/certs/applications/`*CertName*`.p12"];`
	`};`

3	By default, the application will prompt the user for the certificate pass phrase as it starts up. To choose another option for providing the pass phrase, see "Providing a Certificate Pass Phrase" on page 368.

Step	Action
4	If you are using the KDM to enable automatic activation of your secure servers, make sure you update the KDM database with the new certificate passwords. See “Automatic Activation of Secure Servers” on page 381.

Deploying Certificates in Smart Cards

Overview

Orbix supports an option to store credentials (that is, an X.509 certificate chain and private key) on a smart card.

Prerequisites

Before deploying your certificates in a smart card, you must have the following third-party products installed:

- *Baltimore smart card toolkit*—a software library that supports the PKCS#11 interface and enables Orbix to communicate with the smart card (see http://www.baltimore.com). This library is bundled with Orbix.
- Tools and utilities to administer the smart card (usually bundled with the hardware).

Deploying the certificates

Smart card hardware is normally delivered with drivers and utilities that enable you to deploy X.509 certificate chains and private keys to the smart card. Consult the *third-party documentation* that accompanies your smart-card hardware for details.

Deployment constraints

Please note the following constraints when deploying the certificates:

- You must deploy the certificate chain and private key to slot `0`. This is currently the only supported smart card slot.
- The slot `0` should contain only *one* certificate chain and public/private key pair.

Configuring an application to use the smart card

To configure an application to use the smart card, edit the configuration for your domain (usually *ASPInstallDir*`/etc/domains/`*DomainName*`.cfg`). In the *SmartCardApp* scope, ensure that the principal sponsor is configured to use the smart card, as follows:

```
# Orbix Configuration File
SmartCardApp {
  ...
  principal_sponsor:use_principal_sponsor = "true";
  principal_sponsor:auth_method_id = "pkcs11";
```

```
  principal_sponsor:auth_method_data = ["provider=dkck132.dll",
    "slot=0"];
};
```

By default, the application will prompt the user for the smart card PIN as it starts up. To choose another option for providing the PIN, see “Providing a Smart Card PIN” on page 372.

Deploying Orbix Service Certificates

Orbix services requiring certificates

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

- Locator,
- Node daemon,
- Naming service,
- Interface repository (IFR),
- Management service.
- Security service.

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

Choosing a deployment directory

Orbix service certificates are stored as PKCS#12 files. The service certificates are similar to application certificates and, like application certificates, can be stored in arbitrary locations. It is usually convenient to store the service certificates in their own subdirectory—for example:

X509Deploy`/certs/services`

Deploying

To deploy a service certificate, *CertName*`.p12`, for a service that uses the *Service* ORB name in the *DomainName* domain, perform the following steps:

Step	Action
1	Copy the service certificate, *CertName*`.p12`, to the service certificates directory *X509Deploy*`/certs/services` on the deployment host. The service certificates directory should be a secure directory that is accessible only to administrators and other privileged users.

Step	Action

2	Edit the *DomainName* configuration file (usually *ASPInstallDir*/etc/domains/*DomainName*.cfg). In the *Service* scope, change the principal sponsor configuration to specify the *CertName*.p12 certificate, as follows:
	`# Orbix Configuration File`
	`Service {`
	`  ...`
	`  principal_sponsor:use_principal_sponsor = "true";`
	`  principal_sponsor:auth_method_id = "pkcs12_file";`
	`  principal_sponsor:auth_method_data =`
	`   ["filename=X509Deploy/certs/services/CertName.p12`
	`   "];`
	`};`

3	By default, the application will prompt the user for the certificate pass phrase as it starts up. To choose another option for providing the pass phrase, see "Providing a Certificate Pass Phrase" on page 368.
4	If you are using the KDM to enable automatic activation of the Orbix service, make sure you update the KDM database with the new certificate pass phrase. See "Automatic Activation of Secure Servers" on page 381.

Providing pass phrases for Orbix services

It is possible to combine the different ways of providing pass phrases to the Orbix services. For example, some of the alternatives for setting up the Orbix services are:

- Use a password file for all Orbix services.
- Provide the pass phrase from a dialog prompt for all Orbix services.
- Use a password file for the locator and the node daemon. Use the KDM for all other Orbix services.
- Provide the pass phrase from a dialog prompt for the locator and the node daemon. Use the KDM for all other Orbix services.

Example configuration

The default configuration of the Orbix services specifies that all services use the `administrator.p12` certificate. The principal sponsor for services is configured as follows:

```
# Orbix Configuration File
iona_services
{
  principal_sponsor:use_principal_sponsor = "true";
  principal_sponsor:auth_method_id = "pkcs12_file";
  principal_sponsor:auth_method_data =
   ["filename=ASPInstallDir\asp\6.0\etc\tls\x509\certs\services\a
   dministrator.p12",
   "password_file=ASPInstallDir\asp\6.0\etc\tls\x509\certs\servic
   es\administrator.pwf"];
  ...
  ServiceA {
    // Inherit principal sponsor settings from outer scope.
    ...
  };
  ServiceB {
    // Inherit principal sponsor settings from outer scope.
    ...
  };
  ...
};
```

The sub-scopes, *ServiceA*, *ServiceB* and so on, use the principal sponsor settings from the outer scope, `iona_services`. Hence, all of the Orbix services use the same certificate, `administrator.p12`.

It is possible to override settings from the `iona_services` outer scope by configuring the principal sponsor in a local scope—for example, within the *ServiceA* scope.

Deploying itadmin Certificates

Overview

The Orbix command-line administration utility, `itadmin`, requires a certificate when used in a secure domain. Two categories of certificate can be used with `itadmin`, as follows:

- *Ordinary certificates*—for users with ordinary privileges who want to perform routine administration tasks such as checking the status of servers and administering the naming service.
- *Administrator certificates*—for users with administrator privileges who need to administer pass phrases and security checksums stored in the KDM—see "KDM Administration" on page 389.

Specifying a deployment directory for administrator certificates

Before deploying `itadmin` certificates for the first time, you can edit the Orbix configuration file to specify the directory that will contain the administrator certificates. You can specify the administrator certificates deployment directory using the `itadmin_x509_cert_root` configuration variable.

For example, if you choose the following deployment directory for your `itadmin` certificates:

X509Deploy`/certs/admin`

you should then set `itadmin_x509_cert_root` as follows:

```
# Orbix Configuration File
itadmin_x509_cert_root = "X509Deploy/certs/admin";
...
```

Deploying an ordinary certificate for itadmin

To deploy an ordinary certificate for `itadmin`, *OrdinaryCert*`.p12`, in the *DomainName* domain, perform the following steps:

Step	Action
1	Copy the ordinary certificate, *OrdinaryCert*`.p12`, to the service certificates directory *X509Deploy*`/certs/services` on the deployment host. The service certificates directory should be a secure directory that is accessible only to administrators and other privileged users.

2	Edit the *DomainName* configuration file (usually *ASPInstallDir*`/etc/domains/`*DomainName*`.cfg`). In the *ItadminUtility* scope, change the principal sponsor configuration to specify the *OrdinaryCert*`.p12` certificate, as follows:
	`# Orbix Configuration File`
	ItadminUtility `{`
	`  ...`
	`  principal_sponsor:use_principal_sponsor = "true";`
	`  principal_sponsor:auth_method_id = "pkcs12_file";`
	`  principal_sponsor:auth_method_data =`
	`   ["filename=`*X509Deploy*`/certs/services/`*OrdinaryCert*`.p`
	`   12"];`
	`};`

3	By default, the `itadmin` utility would prompt the user for the certificate pass phrase as it starts up. A more convenient option, however, is to store the pass phrase in a secure password file—see "Providing a Certificate Pass Phrase" on page 368 for details of how to configure this.

Deploying an administrator certificate for itadmin

To deploy an administrator certificate for `itadmin`, *AdminCert*`.p12`, perform the following step:

Step	Action
1	Copy the administrator certificate, *AdminCert*`.p12`, to the `itadmin` certificates directory specified by the `itadmin_x509_cert_root` configuration variable. The `itadmin` certificates directory should be a secure directory that is accessible only to administrators and other privileged users.

Overriding the ordinary certificate with the administrator certificate

To perform administrator tasks requiring special privileges, such as administering the KDM, you must override the ordinary certificate with the administrator certificate using the `itadmin admin_logon` subcommand.

See "KDM Administration" on page 389 for details.

Configuring Certificate Warnings

Overview

Orbix enables you to configure the following kinds of certificate warning:

- Certificate expiration warning.
- Own credentials warning.

Certificate expiration warning

Normally, an X.509 certificate would be defined to expire after a certain date. You can arrange to send a warning message to the Orbix log, if certificate expiration is imminent, thus helping to avoid unexpected failure.

To configure a certificate expiration warning, add the configuration variables from Example 36 to your application's configuration scope.

Example 36: *Configuring a Certificate Expiration Warning*

```
#Orbix Configuration File
plugins:iiop_tls:enable_warning_for_approaching_cert_expiration
   = "true";
plugins:iiop_tls:cert_expiration_warning_days = "31";
```

The configuration in Example 36 would send a warning to the Orbix log, if the application's own certificate is less than 31 days away from expiry. Only an application's own certificate is checked, *not* the peer certificates.

Own credentials warning

You can also configure Orbix to log a warning, if the subject DN from an application's own certificate matches a certain pattern. This can be useful, for example, if you want to ensure that demonstration certificates are not accidentally deployed in a production system.

Example 37 shows how to configure the *own credentials warning*. If the specified certificate constraints match the subject DN of an application's own certificate, a warning is issued to the Orbix log. For details of the constraint language, see "Applying Constraints to Certificates" on page 545.

Example 37: *Configuring an Own Credentials Warning*

```
#Orbix Configuration File
plugins:iiop_tls:own_credentials_warning_cert_constraints =
   ["C=US,ST=Massachusetts"];
```

Deploying Certificates with Schannel

Overview

This section describes how to deploy X.509 certificates into the Schannel certificate store. This method of deployment is used *only* for C++ applications that use the Schannel SSL/TLS toolkit on the Windows platform—see "Choosing an SSL/TLS Toolkit" on page 269 for more details.

In this section

This section contains the following subsections:

Schannel Certificate Store

Overview

This subsection describes how to manage certificates in the Schannel certificate store (Windows C++ applications only).

Prerequisites

The Schannel certificate store is *only* available to C++ applications on the Windows platform when you have selected Schannel as the underlying SSL/TLS toolkit. See "Choosing an SSL/TLS Toolkit" on page 269 for details.

Managing the certificate store

Windows makes the Schannel certificate store accessible through the following O/S utilities:

- Internet Explorer.
- Microsoft Management Console.

Internet Explorer

To access the certificate store from Internet Explorer:

1. Choose the **Tools|Internet Options...** menu option to open the **Internet Options** dialog box.
2. Click on the **Content** tab.
3. Click **Certificates...** to open the **Certificates** dialog box.
4. Use the **Certificates** dialog box to manage the certificate store.

Microsoft Management Console

You can also access the certificate store from within the Microsoft Management Console (MMC), using the *certificate snap-in*. The MMC is general-purpose, customizable management tool for the Windows operating system. The functionality of the MMC can be customized by adding, removing and configuring a variety of different MMC snap-ins.

You can add the certificate snap-in to the MMC as follows:

1. Start the MMC from the start menu by selecting **Start|Run** and then entering the command `mmc`. The MMC opens as shown in Figure 44.

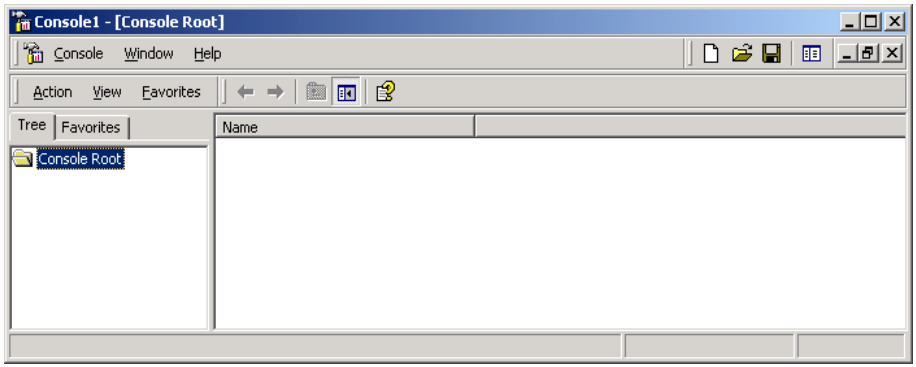

Figure 44: *The Microsoft Management Console*

2. From the MMC, select the **Console|Add/Remove Snap-In...** menu option. The **Add/Remove Snap-In** dialog opens as shown in Figure 45.

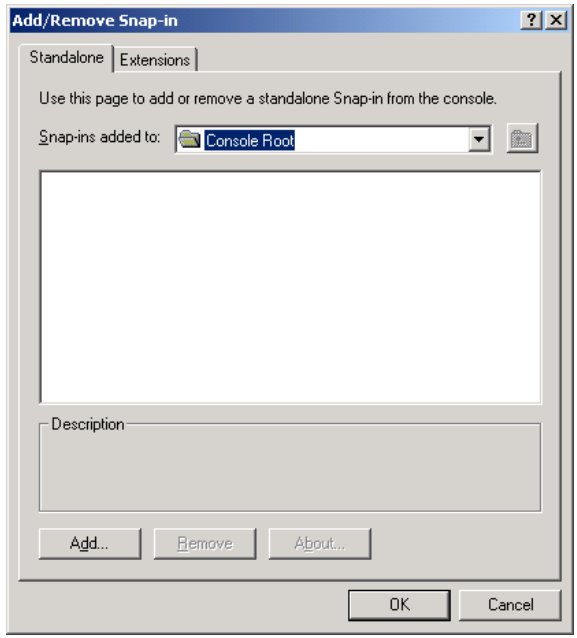

Figure 45: *The Add/Remove Snap-In Dialog Box*

3. Click **Add...** to open the **Add Standalone Snap-In** dialog box, as shown in Figure 46.

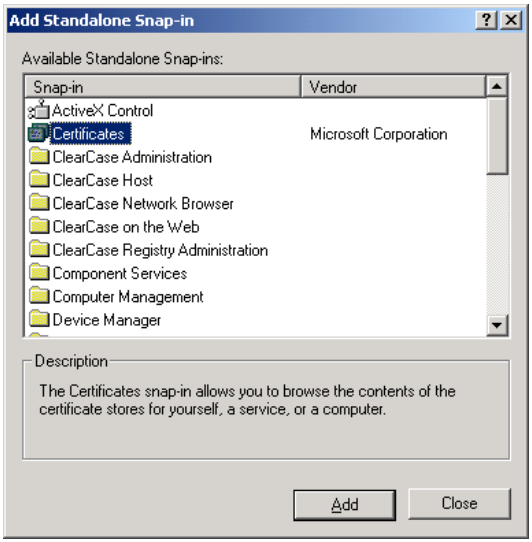

Figure 46: *The Add Standalone Snap-In Dialog Box*

4. From the snap-in list box, select the Certificates snap-in and then click **Add**.
5. A wizard utility starts up to guide you through the process of adding the Certificates snap-in. Follow the instructions in the wizard to add the snap-in.

6. After finishing the certificate snap-in wizard, close the dialog boxes. The console window should now look similar to Figure 47.

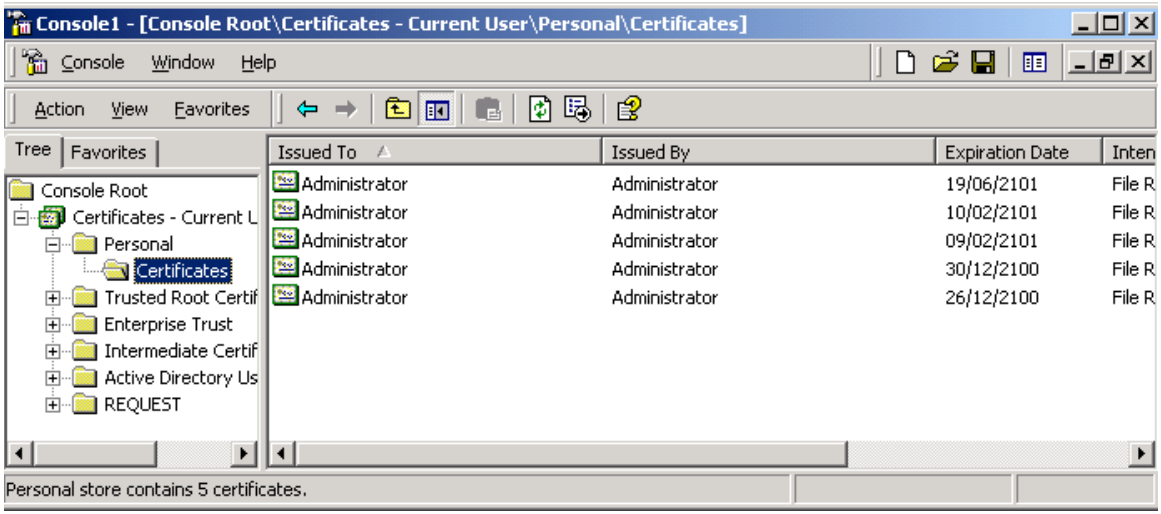

Figure 47: *Microsoft Management Console with Certificates Snap-In*

7. To save the current console configuration for future use, select **Console|Save As...** and save the customized console in a convenient location.

References

For more details about the MMC utility, see the following white paper from Microsoft:

- Microsoft Management Console: Overview (http://www.microsoft.com/windows2000/docs/_Toc463917037).

Deploying Trusted Certificate Authorities

Overview

This subsection describes how to deploy trusted certificate authority (CA) certificates to the Schannel certificate store (Windows C++ applications only). Your Orbix application must be configured to use Schannel as its underlying SSL/TLS toolkit.

CA certificate format

A trusted CA certificate is distributed as a plain certificate without a private key (the private key is known only to the certification authority). For example, trusted CA certificates might be distributed in PEM format, but not in PKCS#12 format (which includes a private key).

Deploying

To deploy a trusted CA certificate to the Schannel certificate store, perform the following steps:

1. Launch an MMC utility that has been configured with a certificates snap-in (see "Schannel Certificate Store" on page 315).
2. From the MMC console tree, select the **Console Root\Certificates\Trusted Root Certification Authorities\Certificates** directory.
3. Right-click the **Certificates** directory and select the **All Tasks|Import...** option. A **Certificate Import Wizard** launches.
4. Follow the instructions in the **Certificate Import Wizard** to add a trusted CA certificate to the certificate store.

Note: The Orbix `policies:iiop_tls:trusted_ca_list_policy` configuration variable is ignored when your C++ application is configured to use the Schannel SSL/TLS toolkit.

Deploying Application Certificates

Overview

This subsection describes how to deploy application certificates in the Schannel certificate store (Windows C++ applications only). Your Orbix application must be configured to use Schannel as its underlying SSL/TLS toolkit.

Deploying

To deploy an application certificate to the Schannel certificate store, perform the following steps:

1. Launch an MMC utility that has been configured with a certificates snap-in (see "Schannel Certificate Store" on page 315).
2. From the MMC console tree, select the **Console Root\Certificates\Personal\Certificates** directory.

   Note: Currently, Orbix can load application certificates from the personal certificate directory only.

3. Right-click the **Certificates** directory and select the **All Tasks|Import...** option. A **Certificate Import Wizard** launches.
4. Follow the instructions in the **Certificate Import Wizard** to add an application certificate to your personal certificate store.
5. To configure an Orbix application to use the certificate, you need to know the common name (CN) from the certificate's subject DN.

   If you do not already know the certificate's common name, you can easily find out by double-clicking the certificate entry in the **Console Root\Certificates\Personal\Certificates** directory of the MMC console. In the **Certificate** dialog, click the **Details** tab and then select the

Subject field from the scrollbox. Figure 48 shows the Certificate dialog at this point.

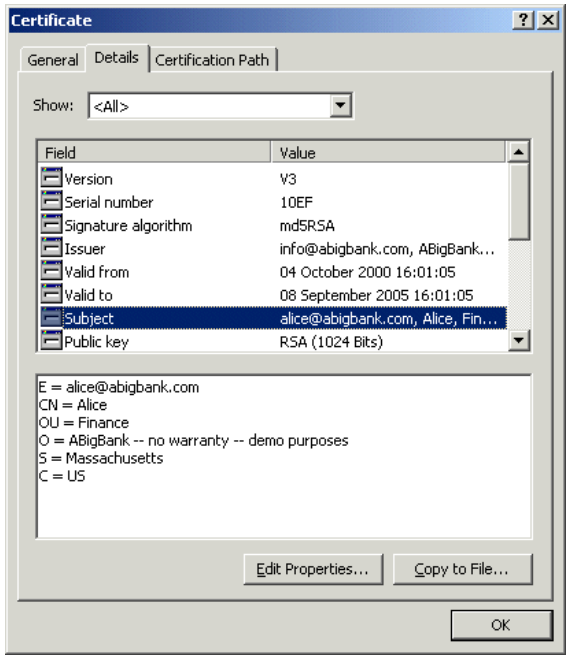

Figure 48: *Certificate Dialog Showing the Certificate's Subject DN.*

The lower pane shows the AVA settings from the certificate's subject DN (for an explanation of X.509 certificate terminology, see "ASN.1 and Distinguished Names" on page 629). From Figure 48, you can see that the common name (CN) of this certificate is `Alice`.

6. Edit the Orbix configuration for your domain (usually *ASPInstallDir*`/etc/domains/`*DomainName*`.cfg`). In your application's configuration scope, *MyApp*, ensure that the principal sponsor is configured to use the new certificate, as shown in Example 38.

```
# Orbix Configuration File
...
MyApp {
  ...
  principal_sponsor:use_principal_sponsor = "true";
  principal_sponsor:auth_method_id = "security_label";
  principal_sponsor:auth_method_data =
   ["label=CommonName"];
};
```

Where *CommonName* is the common name (CN) from the new certificate's subject DN. For example, if using the certificate shown in Figure 48 on page 322, the *CommonName* would be `Alice`.

Note: When Orbix is configured to use Schannel, you cannot use PKCS#12 files directly. Hence, the `pkcs12_file` value of `principal_sponsor:auth_method_id` cannot be used with Schannel.

7. When you start an Orbix application that uses the new certificate, Schannel might or might not prompt you for a private key password. The behavior at runtime depends on whether or not you chose the **Enable strong private key protection** option when importing the certificate with the **Certificate Import Wizard**.

Importing PKCS#12 files

If you want to import a PKCS#12 certificate (`.p12` file suffix) into the certificate store, there is an easy short cut available: double-click the PKCS#12 file and follow the instructions in the **Certificate Import Wizard** to add the certificate to your personal certificate store.

Deploying Certificates in Smart Cards

Overview

Orbix supports an option to store credentials (that is, an X.509 certificate chain and private key) on a smart card.

This subsection describes how to deploy certificates in a smart card which is accessible through the Schannel certificate store (Windows C++ applications only). Your Orbix application must be configured to use Schannel as its underlying SSL/TLS toolkit.

Prerequisites

Before deploying your certificates in a smart card, you must have the following third-party products installed:

- *Third-party smart card toolkit*—a software library that integrates the smart card hardware with the Schannel toolkit and certificate store.
- Tools and utilities to administer the smart card (usually bundled with the hardware).

Deploying the certificates

Smart card hardware is normally delivered with drivers and utilities that enable you to deploy X.509 certificate chains and private keys to the smart card. Consult the *third-party documentation* that accompanies your smart-card hardware for details.

Smart card transparency in Schannel

As soon as a smart card is inserted into the card reader, the smart card credentials automatically appear in the Schannel certificate store. The credentials are then accessible in just the same way as any other certificate in the store.

Configuring an application to use the smart card

To configure an Orbix application to use the smart card through Schannel, edit the configuration for your domain (usually *ASPInstallDir*`/etc/domains/`*DomainName*`.cfg`). In your application's

configuration scope, *SmartCardApp*, ensure that the principal sponsor is configured to use the smart card, as shown in Example 38.

Example 38: *Configuring an Application to Use a Smart Card in Schannel*

```
# Orbix Configuration File
...
SmartCardApp {
  ...
  principal_sponsor:use_principal_sponsor = "true";
  principal_sponsor:auth_method_id = "security_label";
  principal_sponsor:auth_method_data = ["label=CommonName"];
};
```

Where *CommonName* is the common name (CN) from the smart card certificate's subject DN (see "ASN.1 and Distinguished Names" on page 629).

Supplying the smart card PIN

By default, Schannel will prompt the user for the smart card PIN as it starts up. There is currently no alternative to supplying the smart card PIN in Schannel.

CHAPTER 12

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 334 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 336 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 343 for details.

Figure 49 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.

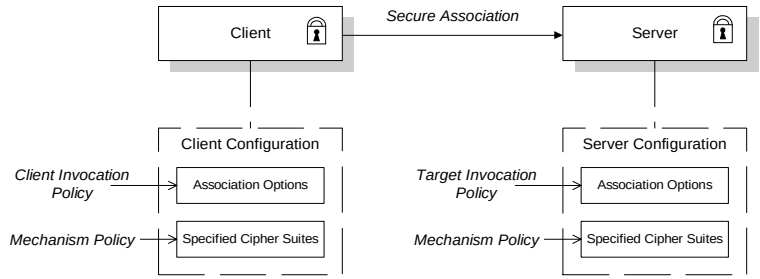

Figure 49: *Configuration of a Secure Association*

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.

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 332 and Appendix D on page 635.

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 332 and Appendix D on page 635.

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 66.

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 348).
- 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 16.

Table 16: *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 17.
- `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 17.

Table 17: *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" on page 570.

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 21 on page 349). As a rule of thumb, if you want security you generally would want *all* of these association options.

Table 18: *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 18.

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

Table 19: *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 66.
- 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_DES40_CBC_SHA
  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 20: *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_DES40_CBC_SHA	RSA_EXPORT	DES40_CBC	SHA	*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, TLS or both are used with a transport protocol by assigning a list of protocol versions to the `policies:iiop_tls:mechanism_policy:protocol_version` configuration variable for IIOP/TLS and the `policies:https:mechanism_policy:protocol_version` configuration variable for HTTPS. For example:

```
# Artix Configuration File
policies:iiop_tls:mechanism_policy:protocol_version = ["TLS_V1",
   "SSL_V3"];
```

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

```
TLS_V1
SSL_V3
```

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

Interoperating with CORBA applications on OS/390

There are some implementations of SSL/TLS on the OS/390 platform that erroneously send SSL V2 client hellos at the start of an SSL V3 or TLS V1 handshake. If you need to interoperate with a CORBA application running on OS/390, you can configure Artix to accept SSL V2 client hellos using the `policies:iiop_tls:mechanism_policy:accept_v2_hellos` configuration variable for IIOP/TLS. For example:

```
# Artix Configuration File
policies:iiop_tls:mechanism_policy:accept_v2_hellos = "true";
```

The default is `false`.

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_DES40_CBC_SHA
  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 50 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*.

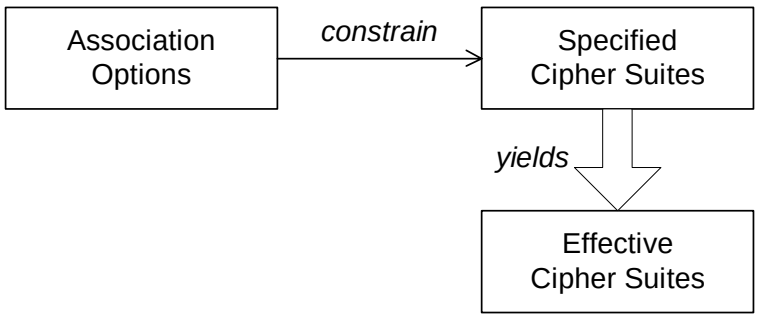

Figure 50: *Constraining the List of Cipher Suites*

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

Table 21: *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_DES40_CBC_SHA	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 21) do not satisfy the configured required association options.
3. From the remaining list, remove any cipher suite that supports an option (see Table 21) 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.

Caching TLS Sessions

Session caching policy

You can use the `IT_TLS_API::SessionCachingPolicy` to control TLS session caching and reuse for both the client side and the server side.

Configuration variable

You can set the `IT_TLS_API::SessionCachingPolicy` with the `policies:iiop_tls:session_caching_policy` or `policies:https:session_caching_policy` configuration variables. For example:

```
policies:iiop_tls:session_caching_policy = "CACHE_CLIENT";
```

Valid values

You can apply the following values to the session caching policy:

```
CACHE_NONE,
CACHE_CLIENT,
CACHE_SERVER,
CACHE_SERVER_AND_CLIENT
```

Default value

The default value is `CACHE_NONE`.

Configuration variable

`plugins:atli_tls_tcp:session_cache_validity_period`

This allows control over the period of time that SSL/TLS session caches are valid for.

Valid values

`session_cache_validity_period` is specified in seconds.

Default value

The default value is 1 day.

Configuration variable

`plugins:atli_tls_tcp:session_cache_size`

`session_cache_size` is the maximum number of SSL/TLS sessions that are cached before sessions are flushed from the cache.

Default value

This defaults to no limit specified for C++.

This defaults to 100 for Java.

CHAPTER 13

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 327. 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 51.

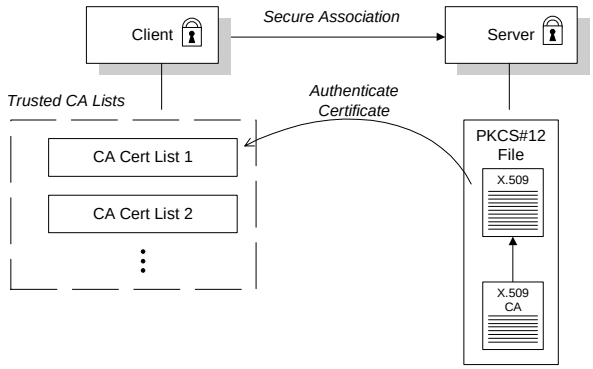

Figure 51: *Target Authentication Only*

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 363.
- One or more lists of trusted certification authorities (CA) are made available to the client—see "Providing a List of Trusted Certificate Authorities" on page 301.

During the security handshake, the server sends its certificate chain to the client—see Figure 51. 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.

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

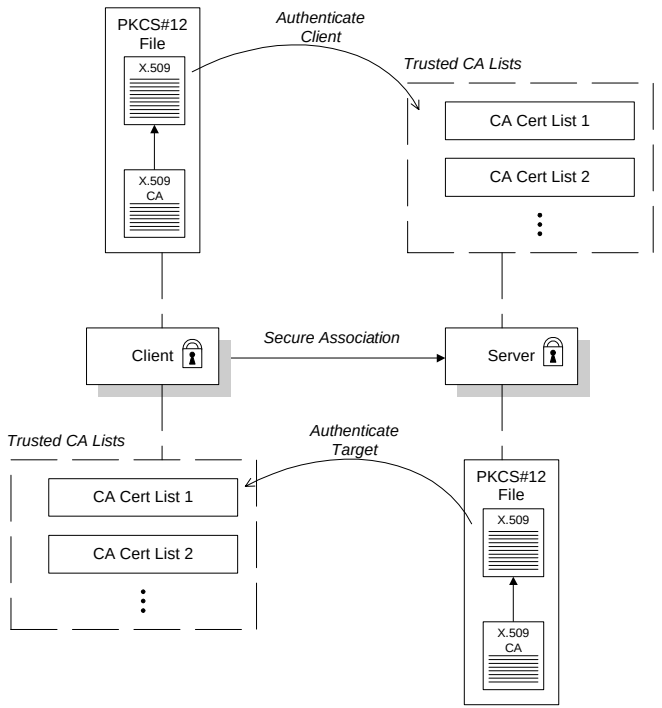

Figure 52: *Target and Client Authentication*

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 363.
- Both client and server are configured with lists of trusted certification authorities (CA)—see "Providing a List of Trusted Certificate Authorities" on page 301.

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 51.

Client configuration

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

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

Server configuration

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

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

Example of target and client authentication

The following sample extract from an Orbix 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 Trusted CA Certificates

Overview

When an application receives an X.509 certificate during an SSL/TLS handshake, the application decides whether or not to trust the received certificate by checking whether the issuer CA is one of a pre-defined set of trusted CA certificates. If the received X.509 certificate is validly signed by one of the application's trusted CA certificates, the certificate is deemed trustworthy; otherwise, it is rejected.

Which applications need to specify trusted CA certificates?

Any application that is likely to receive an X.509 certificate as part of an SSL/TLS or HTTPS handshake must specify a list of trusted CA certificates. For example, this includes the following types of application:

- All IIOP/TLS or HTTPS clients.
- Any IIOP/TLS or HTTPS servers that support mutual authentication.

Deploying trusted CA certificates

You can use one of the following approaches to deploying trusted CA certificates, depending on which SSL/TLS toolkit your application uses:

- Baltimore toolkit (all platforms)—use the Trusted CA list policy.
- Schannel toolkit (Windows C++ applications only)—use the Schannel certificate store.

Trusted CA list policy

The trusted CA list policy specifies a list of files, each of which contains a concatenated list of CA certificates in PEM format. You can configure this policy by setting one of the following configuration variables in your application's configuration scope:

- `policies:iiop_tls:trusted_ca_list_policy`, for IIOP/TLS, and
- `policies:https:trusted_ca_list_policy`, for HTTPS.

Schannel certificate store

If you have configured your application to use the Schannel SSL/TLS toolkit (Windows C++ applications only), you would deploy trusted CA certificates by adding them to the Schannel certificate store, which is an integral part of the Windows operating system.

More details

For more details about deploying trusted CA certificates, see one of the following references:

- Baltimore toolkit—"Providing a List of Trusted Certificate Authorities" on page 301.
- Schannel toolkit—"Deploying Trusted Certificate Authorities" on page 320.

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 461.

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 284).
- 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 53 shows the typical elements in a PKCS#12 file.

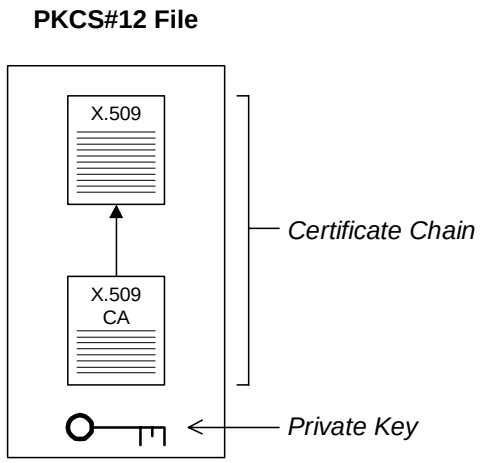

Figure 53: *Elements in a PKCS#12 File*

PKCS#11 and smart cards

Orbix supports the use of smart cards for storing credentials. Orbix accesses the smart card through a standard PKCS#11 interface (implemented by the third-party toolkit from Baltimore).

Smart card storage is arranged as a series of *slots*. To use the smart card with Orbix, slot `0` should be initialized to contain an X.509 certificate chain and a public/private key pair. The user gains access to the data in the smart card by supplying a slot number and a PIN.

Schannel certificate store

(Windows C++ applications only) If you have configured your application to use the Schannel toolkit, the applications own certificate will be stored in the *Schannel certificate store*, which is an integral part of the Windows operation system. For details of how to manage the certificate store, see "Schannel Certificate Store" on page 315.

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 461). If an application is programmed to own multiple certificates, that application ought to be accompanied by documentation that explains how to specify the certificates.

Principal sponsor configuration

To use a principal sponsor, you must set the `principal_sponsor` configuration variables:

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.

Sample PKCS #12 configuration

For example, to use a certificate, *DemoCerts*`/demo_cert_ie5.p12`, that has its password in the *DemoCerts*`/demo_cert_ie5.pwf` file:

```
principal_sponsor:use_principal_sponsor = "true";
principal_sponsor:auth_method_id = "pkcs12_file";
principal_sponsor:auth_method_data =
   ["filename=DemoCerts/demo_cert_ie5.p12",
   "password_file=DemoCerts/demo_cert_ie5.pwf"];
```

Details of these configuration variables can be found in "principal_sponsor Namespace" on page 507.

Sample PKCS #11 configuration

(Java only.) For example, to use a smart card from the provider, `dkck132.dll` (Baltimore), with credentials in slot `0`:

```
principal_sponsor:use_principal_sponsor = "true";
principal_sponsor:auth_method_id = "pkcs11";
principal_sponsor:auth_method_data = ["provider=dkck132.dll",
   "slot=0"];
```

Details of these configuration variables can be found in "principal_sponsor Namespace" on page 507.

Sample Schannel configuration

(Windows C++ applications only) If you have configured your application to use the Schannel toolkit, you should set the principal sponsor as follows:

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

`principal_sponsor:auth_method_data = ["label=`*CommonName*`"];`
Where *CommonName* is the common name (CN) from the certificate's subject DN (see "ASN.1 and Distinguished Names" on page 629).

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.

Providing a Pass Phrase or PIN

Overview

When you specify an application's own certificate, in the form of a certificate file or smart card, you must also provide authorization data that decrypts the certificate's private key, as follows:

- PKCS#12 certificate file—provide a pass phrase,
- PKCS#11 or Schannel smart card—provide a PIN.

In this section

This section contains the following subsections:

Providing a Certificate Pass Phrase

Overview

Once you have specified a PKCS#12 certificate, you must also provide its *pass phrase*. The pass phrase is needed to decrypt the certificate's private key (which is used during the TLS security handshake to prove the certificate's authenticity).

The pass phrase can be provided in one of the following ways:

- From a dialog prompt.
- From the KDM server.
- In a password file.
- Directly in configuration.

From a dialog prompt

If the pass phrase is not specified in any other way, Orbix will prompt the user for the pass phrase as the application starts up. This approach is suitable for persistent (that is, manually-launched) servers.

C++ Applications

When a C++ application starts up, the user is prompted for the pass phrase at the command line as follows:

```
Initializing the ORB
Enter password :
```

Java Applications Using PKCS #12

If the Java application uses a PKCS #12 file to store its certificate, the following dialog window pops up to prompt the user for the pass phrase:

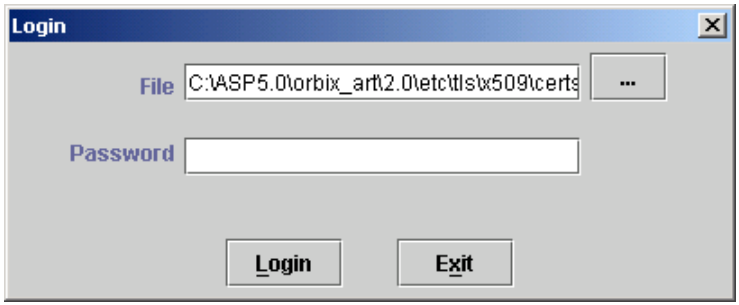

Figure 54: *Java Dialog Window for Certificate Pass Phrase*

The Java dialog window can also be customized by programming. See "principal_sponsor Namespace" on page 507.

From the KDM server

The pass phrase can be obtained automatically from the KDM server as the application starts up. This mechanism is suitable for automatically launched servers. See "Automatic Activation of Secure Servers" on page 381 for details.

In a password file

The pass phrase is stored in a password file whose location is specified in the `principal_sponsor:auth_method_data` configuration variable using the `password_file` option. For example, the `iona_services` scope configures the principal sponsor as follows:

```
# Orbix Configuration File
iona_services {
  ...
  principal_sponsor:use_principal_sponsor = "true";
  principal_sponsor:auth_method_id = "pkcs12_file";
  principal_sponsor:auth_method_data =
   ["filename=ASPInstallDir\asp\6.0\etc\tls\x509\certs\services\a
   dministrator.p12",
   "password_file=ASPInstallDir\asp\6.0\etc\tls\x509\certs\servic
   es\administrator.pwf"];
  ...
};
```

In this example, the pass phrase for the `bank_server.p12` certificate is stored in the `administrator.pwd` file, which contains the following pass phrase:

```
administratorpass
```

WARNING: Because the password file stores the pass phrase in plain text, the password file should not be readable by anyone except the administrator. For greater security, you could supply the pass phrase from a dialog prompt instead.

Directly in configuration

For a PKCS #12 file, the pass phrase can be specified directly in the `principal_sponsor:auth_method_data` configuration variable using the `password` option. For example, the `bank_server` demonstration configures the principal sponsor as follows:

```
# Orbix Configuration File
bank_server {
  ...
  principal_sponsor:use_principal_sponsor = "true";
  principal_sponsor:auth_method_id = "pkcs12_file";
  principal_sponsor:auth_method_data =
   ["filename=ASPInstallDir\asp\6.0\etc\tls\x509\certs\demos\bank
   _server.p12", "password=bankserverpass"];
};
```

In this example, the pass phrase for the `bank_server.p12` certificate is `bankserverpass`.

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

Providing a Smart Card PIN

Overview

If you are using a smart card (PKCS #11 or Schannel), you must provide a PIN when the application starts up to gain access to the smart card.

The PIN can be provided in one of the following ways:

- From a dialog prompt.
- Directly in configuration (PKCS#11 only).

From a dialog prompt

If the PIN is not specified in any other way, Orbix will prompt the user for the PIN as the application starts up.

Java Applications Using PKCS #11 (Smart Card)

If the Java application uses a smart card to store its certificate, the following dialog window pops up to prompt the user for the provider name, slot number, and PIN:

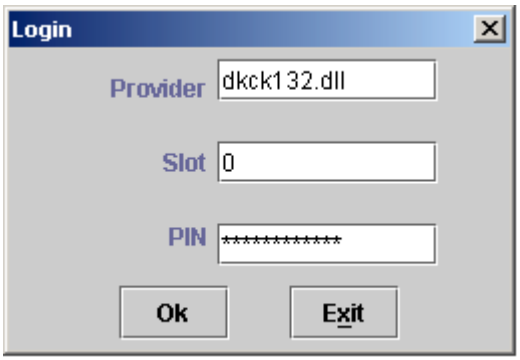

Figure 55: *Java Dialog Window for Certificate PIN*

Windows C++ Application Using Schannel (Smart Card)

If your C++ application is configured to use Schannel in combination with a smart card, the following dialog window pops up to prompt the user for the smart card PIN:

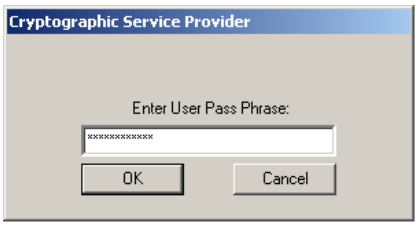

Figure 56: *Schannel Dialog Window for Certificate PIN*

Directly in configuration (PKCS#11 only)

The PKCS #11 authentication mechanism allows you to specify the PIN directly in configuration.

The PIN can be specified directly in the `principal_sponsor:auth_method_data` configuration variable using the `pin` option. For example:

```
# Orbix Configuration File
bank_server {
  ...
  principal_sponsor:use_principal_sponsor = "true";
  principal_sponsor:auth_method_id = "pkcs11";
  principal_sponsor:auth_method_data = ["provider=dkck132.dll",
   "slot=0", "pin=1234"];
};
```

In this example, the PIN for slot `0` of the smart card is `1234`.

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

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 281, 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";
```

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 629.

Delaying Credential Gathering

Overview

Delayed credential gathering is a feature that enables a client to send an X.509 certificate to a secure server at a later point in the SSL/TLS handshake. The advantage of this handshake procedure is that the server sends the client a list of trusted CA certificates. Hence, the client can select a certificate at runtime which is compatible with the server's trusted CA certificates.

Note: Delayed credential gathering is currently *only* supported in combination with the Schannel SSL/TLS toolkit (Windows C++ applications only). See "Choosing an SSL/TLS Toolkit" on page 269.

SSL/TLS handshake process

Delayed credential gathering occurs during the course of the SSL/TLS handshake process as follows:

Stage	Description
1	A client opens a new connection to a secure server and initiates the SSL/TLS connection handshake.
2	The client does *not* initially send an X.509 certificate to the server, although the client supports authentication (that is, the `EstablishTrustInClient` association option is supported on the client side, but the principal sponsor is disabled).
3	At a later stage of the handshake, the server gives the client a second chance to send an X.509 certificate. The server explicitly requests a certificate from the client and sends a list of all the CA certificates it is willing to trust.
4	At this point, if delayed credential gathering is enabled, the client will select a certificate and send it on to the server. Depending on the configuration, the certificate is selected either by default or manually by the user. If delayed credential gathering is *not* enabled, connection establishment would fail at this point.

Enabling delayed credential gathering

Delayed credential gathering is enabled by setting the following variable to true in the relevant scope of your Orbix configuration:

`plugins:iiop_tls:delay_credential_gathering_until_handshake`

When the server requests a client certificate during the SSL/TLS handshake, the certificate can be selected using one of the following procedures:

- Prompting the user for credentials.
- Choosing credentials by default.

Prompting the user for credentials

To enable the user to choose a client certificate at SSL/TLS handshake time, you should set the `plugins:schannel:prompt_with_credential_choice` variable to true. For example:

```
plugins:iiop_tls:delay_credential_gathering_until_handshake =
   "true";
plugins:schannel:prompt_with_credential_choice = "true";
```

Choosing credentials by default

If the `plugins:schannel:prompt_with_credential_choice` variable is set to false, the default behavior is for Orbix to choose the first certificate it can find in the certificate store that meets the applicable constraints. For example, you can enable a default credential choice as follows

```
plugins:iiop_tls:delay_credential_gathering_until_handshake =
   "true";
plugins:schannel:prompt_with_credential_choice = "false";
```

Example client configuration

Example 39 shows how to configure an SSL/TLS client to use delayed credential gathering.

Example 39: *Client Configuration with Delayed Credential Gathering*

```
# Orbix configuration file
...
SchannelClientApplication {
    # Configuration to load Schannel toolkit (not shown)
    ...
    # SSL/TLS Configuration
    policies:client_secure_invocation_policy:requires =
   ["Confidentiality", "EstablishTrustInTarget"];
```

Example 39: *Client Configuration with Delayed Credential Gathering*

```
2     policies:client_secure_invocation_policy:supports =
      ["Confidentiality", "Integrity", "DetectReplay",
      "DetectMisordering", "EstablishTrustInTarget",
      "EstablishTrustInClient"];
      ...
      # Delaying credentials gaterhing
3     principal_sponsor:use_principal_sponsor = "false";
4     plugins:iiop_tls:delay_credential_gathering_until_handshake
      = "true";
5     plugins:schannel:prompt_with_credential_choice = "true";
};
```

The preceding configuration example can be explained as follows:

1. A basic prerequisite for delayed credential gathering is that your application is configured to use the Schannel toolkit (see "Schannel Toolkit for C++" on page 272 for details).
2. The client must support the `EstablishTrustInClient` association option.
3. The principal sponsor *must* be disabled when using the delayed credential gathering feature; in addition you must ensure that no certificate is associated with the client through programming the principal authenticator.
4. The `delay_credential_gathering_until_handshake` variable is set to true to enable delayed credential gathering.
5. In this example, the `prompt_with_credential_choice` variable is set to true so that Schannel will prompt the user for credentials at SSL/TLS handshake time. You could also set this variable to false, if you want to let Orbix choose the credentials by default.

CHAPTER 14

Automatic Activation of Secure Servers

Every server secured with Orbix has an associated certificate and private key. To access its private key, and use it to encrypt messages, a server must retrieve the associated pass phrase. This chapter shows you how to use Orbix administration to supply pass phrases to servers.

In this chapter

This chapter covers the following topics:

Managing Server Pass Phrases

Overview

Every server secured with Orbix has an associated certificate and private key. To access the private key, which is stored in encrypted form, a pass phrase must be supplied to the server as it starts up. The server is then able to identify itself to other applications that require authentication.

Persistent activation

To activate a secure server *persistently* (manual start-up), the server's pass phrase must be supplied by the operator who is starting the process. Typically, the operator types in the pass phrase manually in response to a login prompt at the console.

Automatic activation

To activate a secure server *automatically* (in response to a client request), the server's pass phrase should be supplied automatically because it would be impractical for the server to wait for manual intervention. This is particularly true of high availability environments. It is necessary, therefore, to have a mechanism for automatic delivery of authentication data to a server.

Key distribution management

Orbix provides the *key distribution management* (KDM) mechanism to manage the authentication data required by servers. The KDM manages the storage of authentication data and is responsible for delivering the authentication data to automatically activated servers.

KDM architecture

Figure 57 shows the main components of the KDM architecture:

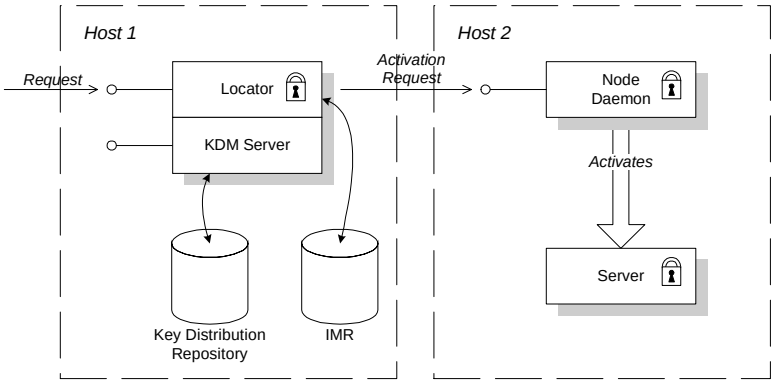

Figure 57: *The KDM Architecture*

The KDM server

The main component of the KDM is the KDM server, which is implemented as a plug-in and embedded in the locator service. The main responsibility of the KDM server is to manage the secure storage and retrieval of authentication data.

The key distribution repository

The *key distribution repository* (KDR) is the database that stores authentication data for the KDM server.The KDR currently stores the following information:

- *Pass phrases*—a pass phrase is stored in the form of an ORB name/pass phrase association. Given an ORB name, the KDM server can retrieve the associated pass phrase. Just one pass phrase can be stored per ORB name.
- *Checksums*—a checksum is generated for a particular server record in the IMR and stored in the form of a process name/checksum association. Checksums are described in "Protecting against Server Imposters" on page 385.

Role of the locator

When the locator receives a client request for an inactive server, the role of the locator is to contact the KDM server (a plug-in to the locator), retrieve the server's authentication data and send the authentication data on to the node daemon.

Role of the node daemon

When the node daemon receives an activation request from the locator, the node daemon launches the corresponding server process and passes the authentication data to the server as it starts up.

Protecting against Server Imposters

Security threats

A server imposter is a rogue server executable that runs in place of a legitimate server application.The KDM must ensure that authentication data are not supplied to server imposters. The following forms of attack must be guarded against:

- Replacing the server executable by an imposter.
- Replacing one or more Orbix plug-ins by imposters.
- Tampering with the IMR record to point at a rogue executable.

Protection measures

The following measures should be taken to protect against server imposters:

- Place all server executables in a trusted directory (for example, one secured by the operating system).
- Place all plug-in libraries in a trusted directory.
- Specify the list of trusted directories in the node daemon's `secure_directories` configuration variable.
- Use the KDM checksum facility to protect the IMR record from tampering.

The secure_directories configuration variable

The `secure_directories` configuration variable specifies a list of trusted directories to the node daemon. For example, on the Windows platform you could set it as follows:

```
# Orbix E2A Configuration File
iona_services {
    ...
    node_daemon {
        secure_directories = ["c:\trusted_servers",
                              "c:\trusted_apps"];
        ...
    };
};
```

If the node daemon's `secure_directories` configuration variable is set, only server executables stored in one of the listed directories can be launched.

Checksums

The server's IMR record contains details of where to find the server executable and other server activation information. By protecting the IMR record from tampering, you can ensure that the KDM passes its authentication data only to a known server executable.

After an administrator creates or modifies a server's IMR record the administrator generates an associated checksum for the IMR record. The checksum is then stored in the KDR database, in the form of a process name/checksum association.

How the KDM Activates a Secure Server

Overview

When the KDM mechanism is used, two different kinds of server activation are supported, as follows:

- *Insecure server activation*—the server is activated using the normal (insecure) activation mechanism. A server is implicitly treated as insecure if no pass phrases are registered for the server.
- *Secure server activation*—the server is activated using a secure activation algorithm. The KDM supplies pass phrases to the server and verifies the server's checksum.

Activation process

Figure 58 outlines the steps for activating a secure server:

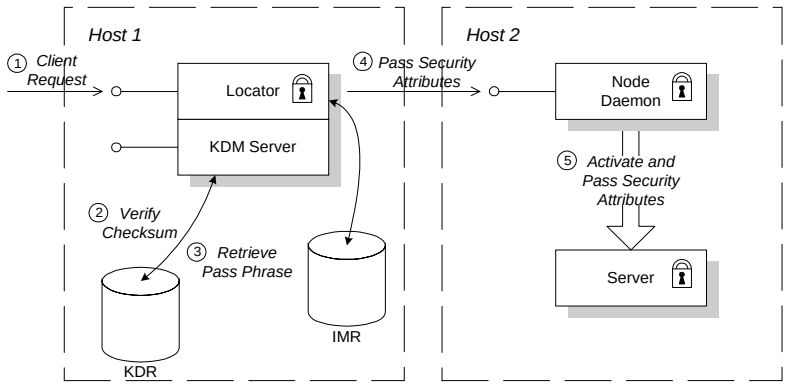

Figure 58: *Automatic Activation of a Secure Server*

Description

The secure server shown in Figure 58 is activated using the KDM, as follows:

Stage	Description
1	A client makes a request on a server that is currently inactive. In Figure 58, the client request (a `Request` or `LocateRequest` message) is sent to the locator. The example assumes that the target object belongs to an indirect persistent POA.
2	The locator requests the server's checksum from the KDM, which attempts to retrieve the checksum from the KDR database. If there is a checksum for the server, the checksum for the server's current IMR record is calculated and compared with the retrieved checksum. If the checksums do not match, the locator reports an error.
3	The locator requests the server pass phrases from the KDM, which retrieves the pass phrases from the KDR database. If there are pass phrases but no checksum for the server, the locator reports an error (unless the `plugins:kdm:checksums_optional` configuration variable is set to `false`). If there are no pass phrases registered for the server, the locator reverts to the standard procedure for activating an insecure server at this point.
4	The locator sends an activation request and authentication data to the node daemon.
5	The node daemon activates the server and passes the authentication data to the server as it starts up.

KDM Administration

Overview

An administrator uses an extended version of the `itadmin` utility to manage the pass phrases and checksums stored in the KDR. In a secure environment, the `itadmin` utility includes a KDM administration plug-in, `kdm_adm`. Figure 59 shows how the `itadmin` utility communicates with the KDM server.

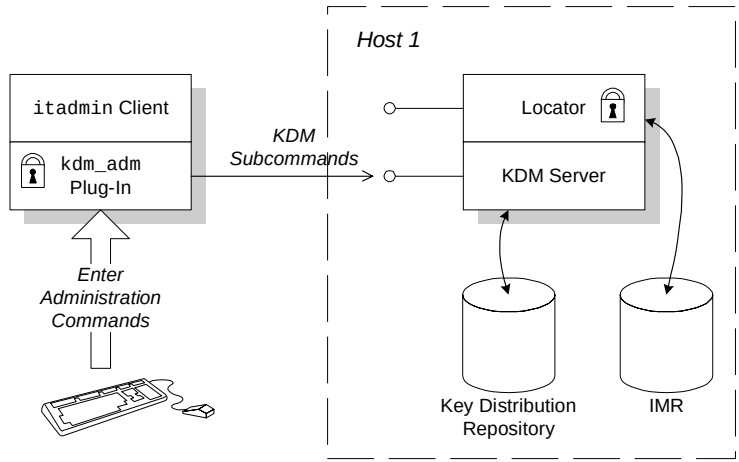

Figure 59: *Using itadmin to Manage the KDM Server*

Whenever the administrator invokes a KDM command (`kdm_adm` or `checksum`) the `itadmin` client communicates directly with a secure IP port on the KDM server (separate from the locator's ports).

Logging In

Before invoking `itadmin` commands to manage the KDM, an administrator must log on to the `itadmin` utility. To log on, enter the following at a command prompt:

```
itadmin
% admin_logon login identity
```

```
Please enter password for identity identity:
%
```

After entering `itadmin`, subsequent commands are entered in `itadmin` script mode (see *Administrator's Guide*). The `admin_logon` command logs the administrator on to the `itadmin` utility using the X.509 certificate specified by *identity*. The administrator then enters the pass phrase to access the certificate.

See the *Administrator's Guide* for full details of the `admin_logon` command syntax.

Commands

Two new administration commands, `kdm_adm` and `checksum`, are provided for the KDM. These commands are used from within the `itadmin` scripting mode.

The `kdm_adm` command manages pass phrases stored in the KDR. The command supports the following subcommands and options:

Table 22: *The kdm_adm Administration Command*

Command	Subcommand and Options
`kdm_adm`	`create -orbname` *name* `[-password` *pass_phrase*`]`
	`confirm -orbname` *name*
	`remove -orbname` *name*
	`list [-count]`
	`change_pw`

The `checksum` command manages server checksums stored in the KDR. The command supports the following subcommands and options:

Table 23: *The checksum Administration Command*

Command	Subcommand and Options
`checksum`	`create -orbname` *name* `[-password` *pass_phrase*`]`
	`confirm -orbname` *name*
	`remove -orbname` *name*
	`list [-count]`

See the *Administrator's Guide* for detailed descriptions of these commands. Examples of using these commands appear in "Registering a Secure Server" on page 394.

Configuration

The KDM is configured by two sets of variables, as follows:

Table 24: *Prefixes for KDM Configuration Variables*

Prefix	Description
`plugins:kdm`	Variables with this prefix configure the KDM server plug-in, which is embedded in the locator service.
`plugins:kdm_adm`	Variables with this prefix configure the KDM administration plug-in, which is embedded in the `itadmin` utility.

A complete list and descriptions of KDM configuration variables is provided in the Appendix A on page 485.

Setting Up the KDM

Setting up a secure domain

Use the `itconfigure` utility to create a secure domain that includes the KDM. You *must* choose file-based configuration instead of the configuration repository (CFR) on a secure domain, because the CFR is completely insecure.

WARNING: Because there is no security on the CFR, anyone could update the CFR so that the KDM uses their certificate. Such an individual would then be able to read all the KDM passwords.

Using secure directories

When an administrator enables automatic activation of a secure server, it becomes possible for remote clients to trigger activation of the secure server. It is, therefore, essential to protect server executables from being overwritten by storing them in a trusted directory.

Create a directory, *SecureServerDir*, that is accessible only to administrators and store your secure server executables in this directory. Add the secure directory, *SecureServerDir*, to the node daemon's list of trusted directories. For example:

```
# Orbix E2A Configuration File
iona_services {
    ...
    node_daemon {
        secure_directories = ["SecureServerDir"];
        ...
    };
};
```

Defining certificate constraints

In a real deployment, you must define a set of certificate constraints for the KDM. The following certificate constraints are relevant to the KDM:

- `plugins:kdm:cert_constraints`—restricts access to the KDM server, protecting it from unauthorized clients. See "plugins:kdm:cert_constraints" on page 494 for details of how to set this variable.

- `plugins:kdm_adm:cert_constraints`—protects the `itadmin` utility from rogue applications that might attempt to impersonate the KDM server. See "plugins:kdm_adm:cert_constraints" on page 495 for details of how to set this variable.

Creating and installing administration certificates

When you create a new set of X.509 certificates for use with Orbix, you need to choose a naming pattern for your Distinguished Names that is compatible with the KDM certificate constraints. In particular, your certificates should satisfy the following conditions:

- The Orbix locator certificate (also used by the KDM server) must satisfy the `plugins:kdm_adm:cert_constraints` certificate constraints.
- Certificates with administrator privileges should satisfy the `plugins:kdm:cert_constraints` certificate constraints.
- Other certificates must not satisfy the KDM certificate constraints.

To deploy the administrator certificates (that is, the certificates used by `itadmin`), create a secure directory *AdminCerts*, copy the administrator certificates to this directory, and set the `itadmin_x509_cert_root` configuration variable equal to *AdminCerts*.

Registering a Secure Server

Server registration steps

You must register the server with the locator daemon to enable it to find the server when requested by a client. To register the server with the locator, perform the following steps:

1. Enter `itadmin`. This starts the Orbix administration command shell, and avoids typing `itadmin` before each command.
2. Register the server's persistent POA name and ORB name with the locator, using the following commands:

   ```
   % orbname create demos.tls.secure_bank_extended_server
   % poa create -replica demos.tls.secure_bank_extended_server
      bank_server_persistent_poa
   ```

   The first command creates an ORB name called `demos.tls.secure_bank_extended_server`. The second creates a POA name called `bank_server_persistent_poa`, and associates it with `demos.tls.secure_bank_extended_server` ORB name, using the `-replica` option. For more details about POA names and ORB names, see the *Administrator's Guide*.
3. Register the server process name with the locator.

   C++ Server

   To register a C++ process name, use the following command:

   UNIX

   ```
   % process create -node_daemon hostname/it_node_daemon
      -pathname
   {install-dir/asp/6.0/demos/tls/secure_bank_extended/
    cxx_server/server} -args "--use_kdm /tmp/bank.ior"
      secure_bank_extended_process
   ```

   Windows

   ```
   % process create -node_daemon hostname/it_node_daemon
      -pathname
   {install-dir\asp\6.0\demos\tls\secure_bank_extended\
    cxx_server\server.exe} -args "--use_kdm C:\temp\bank.ior"
      secure_bank_extended_process
   ```

   Replace *`hostname`* with your machine's DNS name, and replace *`install-dir`* with the location of your Orbix installation (for example,

c:\iona). The -args parameter specifies command-line arguments (for example, the file used to publish the server object reference).

4. Register the server process name with the appropriate ORB name (in this case, demos.tls.secure_bank_extended_server):

```
orbname modify -process secure_bank_extended_process
   demos.tls.secure_bank_extended_server
```

5. From the itadmin command prompt, log on to the itadmin utility:

```
% admin_logon login kdmadmin
Please enter password for identity kdmadmin:
```

This example uses the kdmadmin.p12 certificate which has the password kdmadminpass.

6. Register the server's pass phrase with the KDM:

```
% kdm_adm create -orbname
   demos.tls.secure_bank_extended_server
Please enter password for orb my_orb_name :
```

The secure_bank_extended_server demonstration uses the bankserver.p12 certificate which has the password bankserverpass.

7. Create and store a checksum for the server's IMR record:

```
% checksum create -process secure_bank_extended_process
```

Running the server

After registering the bank server, you must run the bank server once to initialize the bank.ior file containing a persistent object reference. It is only necessary to run the server explicitly once. Subsequently, the node daemon can activate the bank server automatically in response to client requests.

Part IV

CSIv2 Administration

In this part

This part contains the following chapters:

CHAPTER 15

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

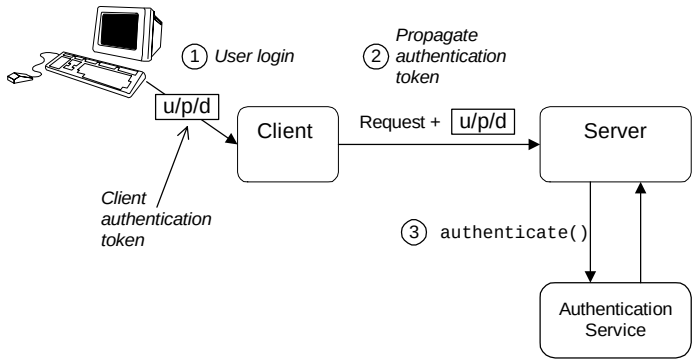

Figure 60: *Basic CSIv2 Authentication over Transport Scenario*

Scenario description

The scenario shown in Figure 60 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 409.

CSIv2 Identity Assertion Scenario

Overview

Figure 61 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.

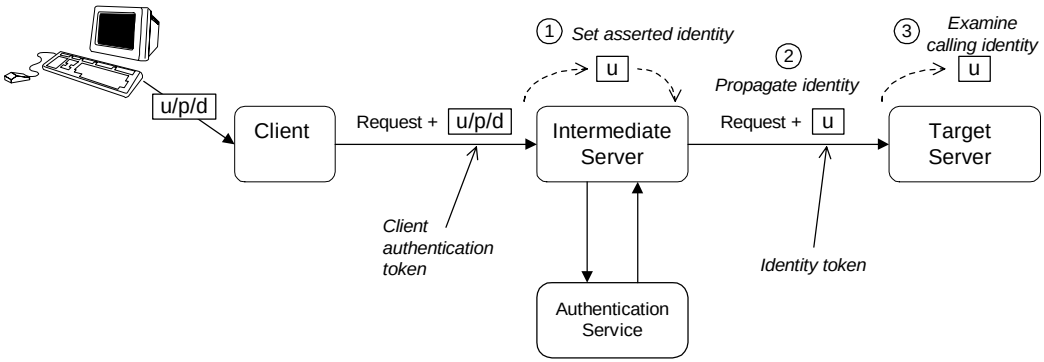

Figure 61: *Basic CSIv2 Identity Assertion Scenario*

Scenario description

The second stage of the scenario shown in Figure 61 (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 429.

Integration with the Orbix Security Framework

Overview

This section presents an example of how CSIv2 works in the context of the Orbix Security Framework. The purpose of the example is to show the distinction between the purely CSIv2 functionality and the way in which CSIv2 is used in the Orbix Security Framework. The example also provides a case study of how to integrate the CSI plug-in within a wider security framework.

CSIv2 authentication domain

In the context of the Orbix Security Framework, the CSIv2 authentication domain set by the user on the client side must match the CSIv2 authentication domain set on the server side.

Plug-ins used by the iSF

Within the iSF, a typical CORBA server would load the following security plug-ins: IIOP/TLS, GSP, and CSI. The roles of the GSP plug-in and the CSI plug-in in particular are important in the context of the iSF, as follows:

- GSP plug-in,
- CSI plug-in.

GSP plug-in

The role of the GSP plug-in is to manage the interpretation of authentication data and to perform authorization. The GSP plug-in implements features specific to the Orbix Security Framework.

CSI plug-in

The role of the CSIv2 plug-in is to manage the propagation of authentication data. It handles the protocol that delivers the data and makes decisions such as whether to propagate authentication data in further calls to other servers.

How CSIv2 integrates with iSF

Figure 62 shows how the CSIv2 and the GSP plug-ins behave in the context of the iSF, for a server that is configured to use CSIv2 authentication over transport.

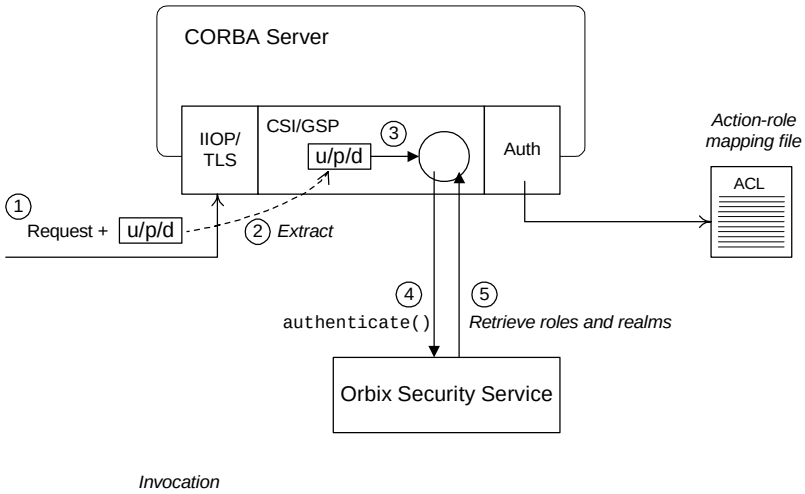

Figure 62: *CSIv2 in the Orbix Security Framework*

Description

The stages of a secure invocation using CSIv2 authentication over transport, as shown in Figure 62, can be described as follows:

Stage	Description
1	A secure operation invocation arrives at the server. Initially, the invocation passes through the IIOP/TLS plug-in, which is responsible for decrypting the incoming message and performing other transport layer security tasks.
2	The CSI plug-in extracts the username/password/domain authentication data, which identifies the calling user, from the incoming message's security attribute service context.

Stage	Description
3	The CSI plug-in delegates authentication to the `IT_CSI::AuthenticateGSSUPCredentials` callback object, which is implemented in the GSP plug-in.
4	The `AuthenticateGSSUPCredentials` object further delegates authentication to the central Orbix security service.
5	If authentication with the Orbix security service is successful, the GSP plug-in receives details of all the roles and realms for the calling user. The roles and realms are cached, to be used later during the authorization step.

CHAPTER 16

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 63 shows a typical scenario for CSIv2 authentication over transport:

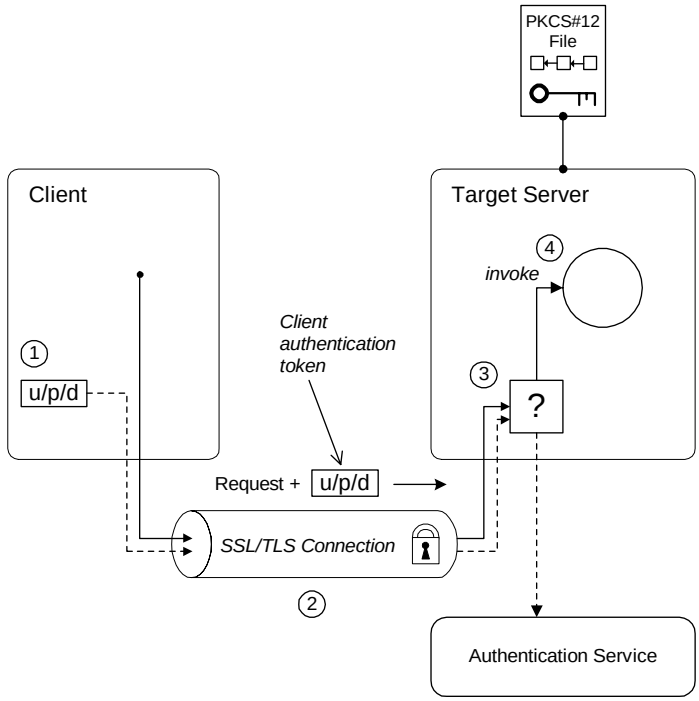

Figure 63: *CSIv2 Authentication Over Transport Scenario*

How CSIv2 authentication over transport proceeds

As shown in Figure 63 on page 411, 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 414 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 420.

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 419.

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 63 on page 411, 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 420). It is possible, therefore, to enable both of the principal sponsors within the same application.

References

See "Sample Configuration" on page 424 for a detailed example of the client and server SSL/TLS configuration.

See "SSL/TLS Administration" on page 267 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 414) and the CSIv2 principal sponsor (see "Providing a Username and Password" on page 420).

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 419 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:

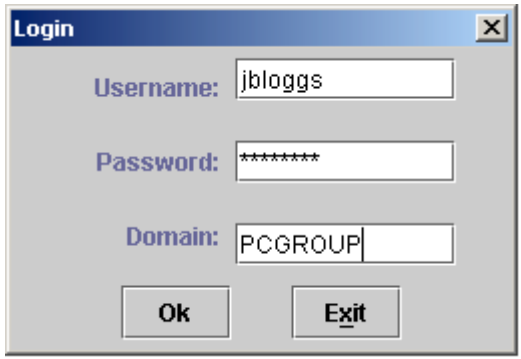

Figure 64: *Java Dialog Window for GSSUP Username and Password*

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 470.

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 410.

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
        {
          plugins:csi:allow_csi_reply_without_service_context =
   "false";
           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 17

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 65). 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 65 shows a typical scenario for CSIv2 identity assertion:

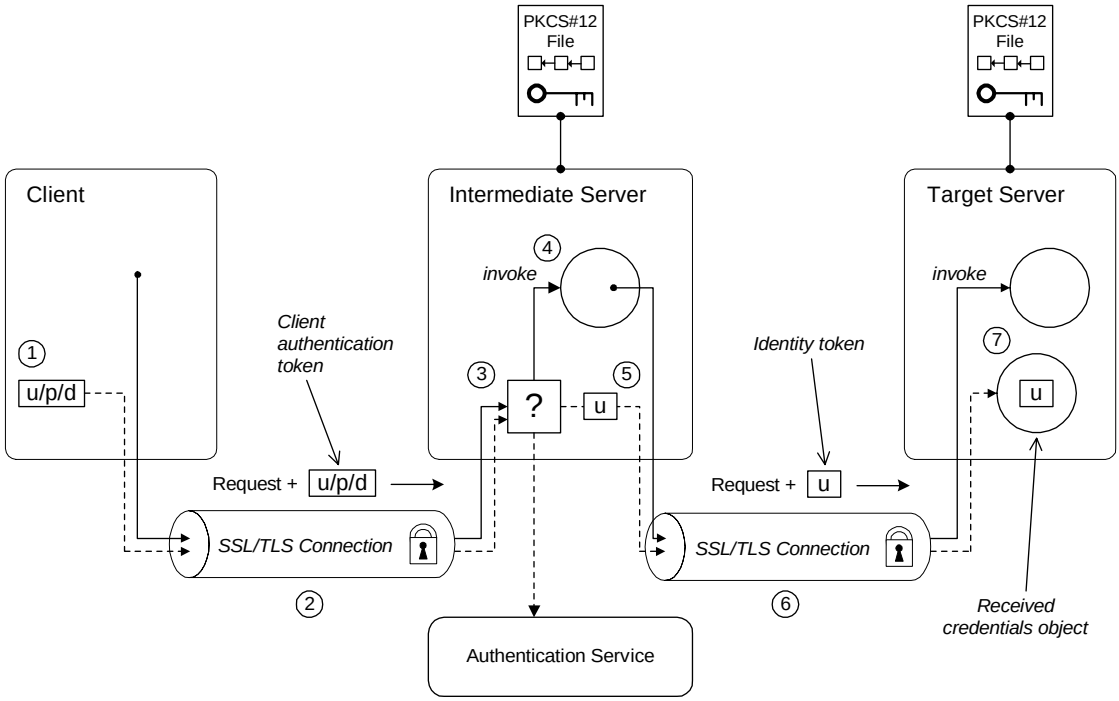

Figure 65: *CSIv2 Identity Assertion Scenario*

How CSIv2 identity assertion proceeds

As shown in Figure 65 on page 431, 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 489.

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 434 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 490.

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 65 on page 431, 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 441 for a detailed example of the SSL/TLS configuration in this scenario.

See "SSL/TLS Administration" on page 267 for complete details of configuring and administering SSL/TLS.

Setting certificate constraints

In the scenario depicted in Figure 65 on page 431, 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 376 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 363 and "Providing a Certificate Pass Phrase" on page 368 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 420). 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 65 on page 431, 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 438.

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 430.

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
        {
          plugins:csi:allow_csi_reply_without_service_context =
   "false";
           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"];

         plugins:csi:allow_csi_reply_without_service_context =
   "false";
         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 18

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 25.

Table 25: *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 40: *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
   );
```

Java Example

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

Example 41: *Java Example of Setting ORB-Level Policies*

```
//Java
1    PolicyManager pol_manager = null;
    pol_manager = (PolicyManager)
      orb.resolve_initial_references("ORBPolicyManager");
    Any policy_value = orb.create_any();
    String[] constraint =
       {"C=US,ST=Massachusetts,O=ABigBank*,OU=Administration"};
2,3 CertConstraintsHelper.insert(policy_value, constraint);
    Policy[] policies = new Policy[1];
```

Example 41: *Java Example of Setting ORB-Level Policies*

```
4,5     policies[0] =
              orb.create_policy(TLS_CERT_CONSTRAINTS_POLICY.value,
              policy_value);
6       pol_manager.set_policy_overrides(policies,
          SetOverrideType.SET_OVERRIDE);
```

Setting a Policy at ORB Level

The programming steps in the preceding examples, "C++ Example" on page 449 and "Java Example" on page 449, 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" on page 570).

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 330, 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 348.

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 348.

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 19

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 363.
- *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_PKCS12_FILE`—enables you to specify the path name of a PKCS#12 file containing an X.509 certificate chain. Not supported by Schannel.
- `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.
- `IT_TLS_API::IT_TLS_AUTH_METH_LABEL`—enables you to specify the common name (CN) from an application certificate's subject DN. This method can be used only in combination with the Schannel toolkit (Windows C++ only).

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 42: *C++ Example of SSL/TLS Authentication*

```
//C++
int pkcs12_login(
     CORBA::ORB_ptr orb,
     const char *pkcs12_filename,
     const char *password
)
```

Example 42: *C++ Example of SSL/TLS Authentication*

```
{
    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;
    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`.

Java example

In the following Java example, a client principal passes its identity to the principal authenticator in the form of a PKCS#12 file:

Example 43: *Java Example of SSL/TLS Authentication*

```
//Java
1    org.omg.SecurityLevel2.SecurityManager manager =
     (org.omg.SecurityLevel2.SecurityManager)
         orb.resolve_initial_references("SecurityManager");

2    PrincipalAuthenticator authenticator
        manager.principal_authenticator();

     Any auth_data_any = orb.create_any();

     PKCS12FileAuthData authentication_data =
        new PKCS12FileAuthData("bankserverpass", certificate);
     PKCS12FileAuthDataHelper.insert(auth_data_any,
        authentication_data);

     SecAttribute[] privileges = new SecAttribute[0];

     // Holder for the credentials returned from logging in
3    CredentialsHolder credentials = new CredentialsHolder();

     // Holders for continuation_data and auth_specific_data
     // are not used
     AnyHolder continuation_data = new AnyHolder();
     AnyHolder auth_specific_data = new AnyHolder();

     AuthenticationStatus authentication_result;
```

Example 43: *Java Example of SSL/TLS Authentication*

```
4    authentication_result = authenticator.authenticate(

     IT_TLS_AUTH_METH_PKCS12_FILE.value,
                                  "",  // mechanism empty
                                  "",  // security name empty
                                  auth_data_any,
                                  privileges,
                                  credentials,
                                  continuation_data,
                                  auth_specific_data
                                );
...
```

Java notes

The preceding Java 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. Initialize an empty credentials holder object to hold the security attributes of this client if login is successful.
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.

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 44 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 44: *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 44: *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 44: *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`.

Java example

Example 45 shows how to create CSIv2 credentials in Java, by supplying a username, *<user_name>*, password, *<password>*, and authentication domain, *<domain>*, to the principal authenticator's `authenticate()` operation.

Example 45: *Java Example of CSIv2 Authentication*

```
//Java
...
// Given the following prerequisites:
// orb - A reference to an org.omg.CORBA.ORB object.

1   org.omg.SecurityLevel2.SecurityManager manager =
    (org.omg.SecurityLevel2.SecurityManager)
        orb.resolve_initial_references("SecurityManager");

2   org.omg.SecurityLevel2.PrincipalAuthenticator authenticator
        = manager.principal_authenticator();
```

Example 45: *Java Example of CSIv2 Authentication*

```
   org.omg.CORBA.Any auth_data_any = orb.create_any();
3  com.iona.IT_CSI.GSSUPAuthData auth_data =
     new com.iona.IT_CSI.GSSUPAuthData(
         "<password>",
         "<domain>"
     );
4   com.iona.IT_CSI.GSSUPAuthDataHelper.insert(
        auth_data_any,
        auth_data
    );

   org.omg.Security.SecAttribute[] privileges
       = new org.omg.Security.SecAttribute[0];

   // Holder for the credentials returned from logging in
5  org.omg.SecurityLevel2.CredentialsHolder credentials
       = new org.omg.SecurityLevel2.CredentialsHolder();

   // Holders for continuation_data and auth_specific_data
   // are not used
   org.omg.CORBA.AnyHolder continuation_data
       = new org.omg.CORBA.AnyHolder();
   org.omg.CORBA.AnyHolder auth_specific_data
       = new org.omg.CORBA.AnyHolder();

   org.omg.Security.AuthenticationStatus authentication_result;

6 authentication_result = principal_authenticator.authenticate(
    com.iona.IT_CSI.IT_CSI_AUTH_METH_USERNAME_PASSWORD.value,
    "",                 // NOT USED
    "<user_name>",  // GSSUP user name
    auth_data_any,      // an any containing the
                        //   IT_CSI::GSSUPAuthData struct
    privileges,         // NOT USED
    credentials,        // returns the CSIv2 user credentials
    continuation_data, // NOT USED
    auth_specific_data // NOT USED
);

// Returned credentials can be accessed in 'credentials.value'
...
```

Java notes

The preceding Java 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. Initialize an empty credentials holder object to hold the security attributes of this client.
6. 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 `credentials.value`.

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 46.

Example 46: *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.

Java example

In Java, you can retrieve an application's own credentials list as shown in Example 47.

Example 47: *Retrieving a Java Application's Own Credentials List*

```
// Java
...
try {
1       org.omg.CORBA.Object obj =
            my_orb.resolve_initial_references("SecurityManager");
        org.omg.SecurityLevel2.SecurityManager security_manager_obj
          = org.omg.SecurityLevel2.SecurityManagerHelper.narrow(obj);
}
catch (org.omg.CORBA.ORB.InvalidName e) {
    ...
}
catch (org.omg.CORBA.BAD_PARAM e)
{
    // Error! Deal with failed narrow...
}

2   org.omg.SecurityLevel2.Credentials[] 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.
    ...
}
```

Java example

In Java, if the own credentials list contains a list of SSL/TLS credentials, you can access the credentials as follows:

```
// Java
import com.iona.corba.IT_TLS_API.TLSCredentials;
import com.iona.corba.IT_TLS_API.TLSCredentialsHelper;
import com.iona.corba.IT_Certificate.X509Cert;
...
for (int i=0; i < creds_list.length; i++)
{
    // Access the i'th own credentials in the list
    TLSCredentials tls_creds =
        TLSCredentialsHelper.narrow(creds_list[i]);

    // Get the first X.509 certificate in the chain
    X509Cert cert =
        tls_creds.get_x509_cert();

    // Examine the X.509 certificate, etc.
    ...
}
```

Parsing CSIv2 Own Credentials

Overview

This subsection explains how to access the information stored in a CSIv2 credentials object. If a credentials object obtained from the security manager is of CSIv2 type, you can narrow the credentials to the `IT_CSI::CSICredentials` type.

Java example

In Java, if the own credentials list contains a list of CSIv2 credentials, you can access the credentials as follows:

```
// Java
import com.iona.corba.IT_CSI.CSICredentials;
import com.iona.corba.IT_CSI.CSICredentialsHelper;
import com.iona.corba.IT_CSI.CSICredentialsType;
import
   com.iona.corba.IT_CSI.CSICredentialsType.GSSUPCredentials;
import
   com.iona.corba.IT_CSI.CSICredentialsType.PropagatedCredential
   s;
...
for (int i=0; i < creds_list.length; i++)
{
    // Access the i'th own credentials in the list
    CSICredentials csi_creds =
        CSICredentialsHelper.narrow(creds_list[i]);
    CSICredentialsType csi_type
        = csi_creds.csi_credentials_type()
    if (csi_type == GSSUPCredentials) {
        System.out.println("[" + i + "] = "
            + "credentials for CSIv2 authentication mechanism");
    }

    ...
}
```

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 48.

Example 48: *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);
...
```

Java example

In Java, you can retrieve the target credentials associated with a particular object reference, `target_ref`, as shown in Example 49.

Example 49: *Java Obtaining Target Credentials*

```
// Java
...
// Given the following prerequisites:
// my_orb - a reference to an ORB instance.
// target_ref - an object reference to a remote, secured object.

try {
    org.omg.CORBA.Object obj =
        my_orb.resolve_initial_references("SecurityManager");
    org.omg.SecurityLevel2.SecurityManager security_manager_obj
      = org.omg.SecurityLevel2.SecurityManagerHelper.narrow(obj);
}
catch (org.omg.CORBA.ORB.InvalidName e) {
    ...
}
catch (org.omg.CORBA.BAD_PARAM e)
{
    // Error! Deal with failed narrow...
}

org.omg.SecurityLevel2.TargetCredentials 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 48 on page 485, 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.
...
```

Java example

In Java, after obtaining a target credentials object, `target_creds`, as shown in Example 49 on page 486, you can access the SSL/TLS specific data as follows (exception handling not shown):

```
// Java
import com.iona.corba.IT_TLS_API.TLSTargetCredentials;
import com.iona.corba.IT_TLS_API.TLSTargetCredentialsHelper;
import com.iona.corba.IT_Certificate.X509Cert;
...
TLSTargetCredentials tls_target_creds =
    TLSTargetCredentialsHelper.narrow(target_creds);

// Get the first X.509 certificate in the chain
X509Cert 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 50.

Example 50: *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();
...
```

Java example

In Java, to obtain received credentials, perform the steps shown in Example 51.

Example 51: *Java Retrieving Received Credentials*

```
// Java
...
// In the context of an operation/attribute implementation

try {
    org.omg.CORBA.Object obj =
        my_orb.resolve_initial_references("SecurityCurrent");
    org.omg.SecurityLevel2.Current current_obj
      = org.omg.SecurityLevel2.CurrentHelper.narrow(obj);
}
catch (org.omg.CORBA.ORB.InvalidName e) {
    ...
}
catch (org.omg.CORBA.BAD_PARAM e)
{
    // Error! Deal with failed narrow...
}

org.omg.SecurityLevel2.ReceivedCredentials 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 50 on page 490) 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.
...
```

Java example

In Java, after obtaining a received credentials object, `recvd_creds`, (see Example 51 on page 491) you can access the SSL/TLS specific data as follows (exception handling not shown):

```
// Java
import com.iona.corba.IT_TLS_API.TLSReceivedCredentials;
import com.iona.corba.IT_TLS_API.TLSReceivedCredentialsHelper;
import com.iona.corba.IT_Certificate.X509Cert;
...
TLSReceivedCredentials tls_recvd_creds =
    TLSReceivedCredentialsHelper.narrow(recvd_creds);

// Get the first X.509 certificate in the chain
X509Cert cert =
    tls_recvd_creds.get_x509_cert();

// Examine the X.509 certificate, etc.
...
```

Parsing CSIv2 Received Credentials

Overview

If you want to access the added value Orbix functionality for CSIv2 received credentials, you need to narrow the generic `SecurityLevel2::ReceivedCredentials` object to the `IT_CSI::CSIReceivedCredentials` type. This subsection explains, with the help of examples, how to access the CSIv2 received credentials.

CSIv2 received credentials

The CSIv2 received credentials are a special case, because the CSIv2 specification allows up to three distinct credentials types to be propagated simultaneously. A CSIv2 received credentials can, therefore, include one or more of the following credentials types:

- Propagated identity credentials (through the CSIv2 identity assertion mechanism).
- GSSUP credentials (through the CSIv2 authentication mechanism).
- Transport credentials (through SSL/TLS).

CSIReceivedCredentials interface

Access to each of the credentials types is provided by the following attributes of the `IT_CSI::CSIReceivedCredentials` interface:

```
// IDL
...
module IT_CSI {
...
    local interface CSIReceivedCredentials :
        IT_TLS_API::TLSReceivedCredentials, CSICredentials
    {
        readonly attribute CSICredentials gssup_credentials;
        readonly attribute CSICredentials
                              propagated_identity_credentials;
        readonly attribute SecurityLevel2::Credentials
                              transport_credentials;
    };
...
};
```

Java example

In Java, after obtaining a received credentials object, `recvd_creds` (see Example 51 on page 491), you can access the CSIv2 specific data as shown in Example 52. This example assumes that CSIv2 authentication is enabled, but not CSIv2 identity assertion. Hence, no attempt is made to access the propagated identity credentials.

Example 52: *Java Parsing CSIv2 Received Credentials*

```
// Java
import org.omg.Security.*;
import org.omg.SecurityLevel2.*;

import com.iona.corba.IT_CSI.CSIReceivedCredentials;
import com.iona.corba.IT_CSI.CSIReceivedCredentialsHelper;
import com.iona.corba.IT_CSI.CSICredentialsType;
import com.iona.corba.IT_CSI.CSI_SERVER_AS_POLICY;
import com.iona.corba.util.OrbServicesUtility;
...
    // Get the TLS received credentials
1   CSIReceivedCredentials csi_rec_creds
        = CSIReceivedCredentialsHelper.narrow(recvd_creds);
2   Credentials transport_credentials_rec
        = csi_rec_creds.transport_credentials();

    // Select the org.omg.Security.AccessId SecAttribute type
3   AttributeType[] attributes_types =
    {
      new AttributeType(
         new ExtensibleFamily((short)0, (short)1), AccessId.value
      )
    };

4   SecAttribute[] trans_attribute
        = transport_credentials_rec.get_attributes(
              attributes_types
          );

5   String trans_access_id = new String(
     trans_attribute[0].value, 0, trans_attribute[0].value.length
    );

    // Get the GSSUP (username/passsword) credentials
6   Credentials gssup_creds = csi_rec_creds.gssup_credentials();

7   SecAttribute[] gssup_attribute
```

Example 52: *Java Parsing CSIv2 Received Credentials*

```
        = gssup_creds.get_attributes(attributes_types);

8   String gssup_access_id = new String(
     gssup_attribute[0].value, 0, gssup_attribute[0].value.length
    );

...
```

The preceding Java example can be explained as follows:

1. This line attempts to narrow the generic received credentials object, `recvd_creds`, to the `IT_CSI::CSIReceivedCredentials` type. If the received credentials object is not of this type, the narrow would fail and a `CORBA::BAD_PARAM` exception would be thrown.
2. The `transport_credentials` attribute accessor returns a reference to the received transport credentials (for example, SSL/TLS), which form part of the overall CSI received credentials. If there is no secure transport or if the client is not configured to send transport credentials, the return value would be `null`.
3. This line initializes a `Security::AttributeTypeList` sequence (Java `org.omg.Security.AttributeType[]` array) with a single attribute type for a `Security::AccessId`.
4. The attribute type list created in the previous line is passed to `get_attributes()` to retrieve the `AccessId` attribute from the received transport credentials. The `AccessId` for the transport credentials is the distinguished name of the subject of the X.509 certificate received from the client. In other words, the `AccessId` identifies the invoking client.
5. This line converts the `AccessId` from its native format (an octet sequence) into a string. The result is a distinguished name in string format (see "ASN.1 and Distinguished Names" on page 629).

   This step completes the process of identifying the client using the transport credentials portion of the CSI received credentials.

6. The `gssup_credentials` attribute accessor returns a reference to the received GSSUP credentials. The GSSUP credentials contain an authenticated username sent by the client using the CSIv2 authentication mechanism. If the client is not configured to use the CSIv2 authentication mechanism, the return value would be `null`.
7. The `get_attributes()` operation is invoked to retrieve the `AccessId` attribute from the received GSSUP credentials. The `AccessId` for the GSSUP credentials is the client's username.
8. This line converts the `AccessId` from its native format (an octet sequence) into a string.

   This step completes the process of identifying the client using the GSSUP portion of the CSI received credentials.

CHAPTER 20

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.

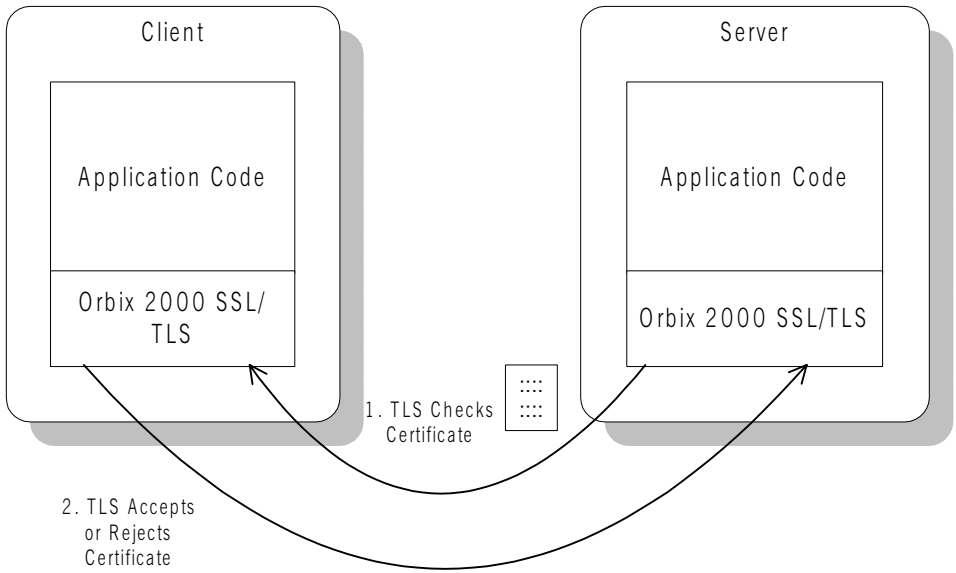

Figure 66: *Validating a Certificate*

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 67 shows the steps followed by Orbix to validate a certificate when a `CertValidatorPolicy` has been registered on the client side:

1. The standard validation checks are applied by Orbix.
2. The certificate is then passed to an `IT_TLS::CertValidator` callback object that performs user-specified validation on the certificate.
3. The user-specified `CertValidator` callback object can decide whether to accept or reject the certificate.

4. Orbix accepts or rejects the certificate.

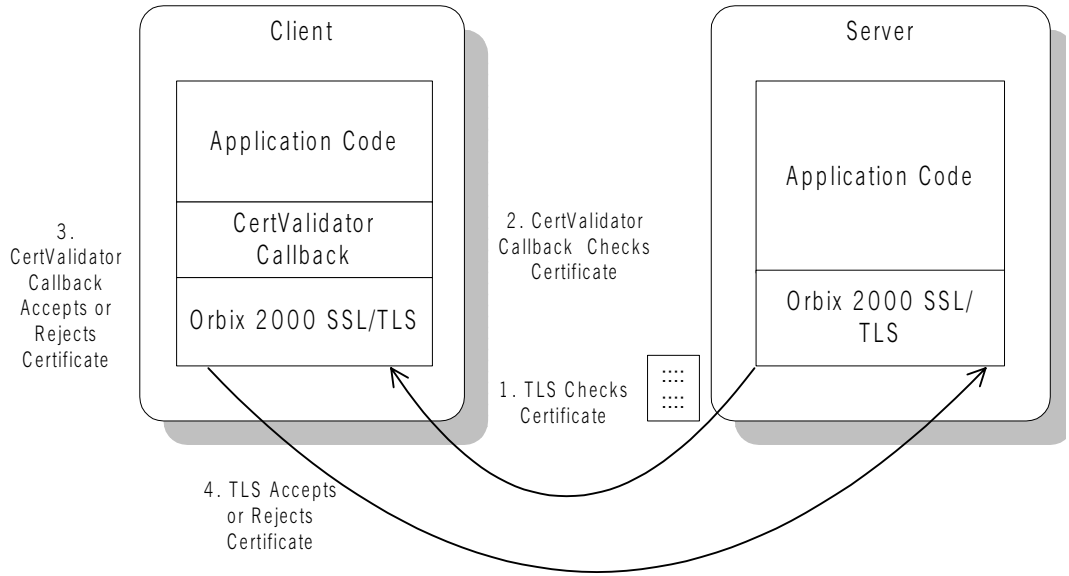

Figure 67: *Using a CertValidator Callback*

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

Parsing APIs

Two distinct APIs are used to parse an X.509 certificate, depending on whether you program in C++ or Java, as follows:

- C++ parsing uses the interfaces defined in the `IT_Certificate` IDL module.
- Java parsing uses the `java.security.cert` package and a subset of the interfaces in the `IT_Certificate` IDL module.

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*.

Java parsing

Orbix E2A ASP allows you to use the X.509 functionality provided by the JDK.

If you develop Java applications, only the following IDL interfaces are relevant:

- `IT_Certificate::Certificate`
- `IT_Certificate::X509Cert`
- `IT_Certificate::X509CertificateFactory`

To access the information in a Java X.509 certificate, perform the following steps:

1. Extract the DER data from the certificate using the `IT_Certificate::Certificate::encoded_form` attribute.
2. Pass the DER data to the `com.iona.corba.tls.cert.CertHelper.bytearray_to_cert()` method to obtain a `java.security.cert.Certificate` object.
3. Use the `java.security.cert` package to examine the certificate.

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 376.

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 53: *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

Java example

The following Java example shows how to set the `CertConstraintsPolicy` programmatically:

Example 54: *Java Example of Setting the CertConstraintsPolicy (Sheet 1 of 2)*

```
// Java
...
// OMG imports
import org.omg.CORBA.ORBPackage.InvalidName;
import org.omg.CORBA.Policy;
import org.omg.CORBA.PolicyManager;
import org.omg.CORBA.PolicyManagerHelper;
import org.omg.CORBA.SetOverrideType;
...
// IONA specific security imports
import com.iona.corba.IT_TLS_API.CertConstraintsHelper;
import com.iona.corba.IT_TLS_API.TLS_CERT_CONSTRAINTS_POLICY;

public class Server
{
  public static void main(String args[])
  {
    try
    {
      ...
      PolicyManager pol_manager = null;
      try
      {
1       pol_manager = PolicyManagerHelper.narrow(
            orb.resolve_initial_references("ORBPolicyManager")
        );
      }
      catch(InvalidName invalid_name)
      {
        System.err.println(
            "x509 initial reference not set. Check plugin list"
```

Example 54: *Java Example of Setting the CertConstraintsPolicy (Sheet 2 of 2)*

```
        );
        System.exit(1);
      }
      catch(org.omg.CORBA.BAD_PARAM exc)
      {
        System.err.println("narrow to PolicyManager failed.");
        System.exit(1);
      }

      org.omg.CORBA.Any policy_value = orb.create_any();
2     String[] constraint =
    {"C=US,ST=Massachusetts,O=ABigBank*,OU=Administration"};
3     CertConstraintsHelper.insert(policy_value, constraint);
      Policy[] policies = new Policy[1];
4     policies[0] = orb.create_policy(
          TLS_CERT_CONSTRAINTS_POLICY.value,
          policy_value
      );
5     pol_manager.set_policy_overrides(
          policies,
          SetOverrideType.SET_OVERRIDE
      );
```

Java example description

The preceding Java example can be explained as follows:

1. Retrieve the `PolicyManager` object.
2. Instantiate a `CertConstraints` data instance (string array).
3. Insert the constraint into `policy_value` (an `Any`).
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 55: *C++ Example of Overriding validate_cert_chain()*

```
//C++
CORBA::Boolean
CustomCertValidatorImpl::validate_cert_chain(
   CORBA::Boolean chain_is_valid,
   const IT_Certificate::X509CertChain& cert_chain,
   const IT_TLS::CertChainErrorInfo& error_info
)
{
    if (chain_is_valid)
    {
       CORBA::String_var CN;
1      IT_Certificate::X509Cert_var cert = cert_chain[0];

2      IT_Certificate::AVAList_var subject =
          cert->get_subject_avalist();

       IT_Certificate::Bytes* subject_string_name;
3      subject_string_name = subject->convert(IT_Certificate::
          IT_FMT_STRING);

       int len = subject_string_name->length();
       char *str_name = new char[len];
       for (int i = 0; i < len; i++){
          str_name[i] = (char)((*subject_string_name)[i]);
       }
    }
    return chain_is_valid;
}
```

The preceding C++ example can be explained as follows:

1. The certificate is retrieved from the certificate chain.
2. An AVAList (see "Working with distinguished names in C++" on page 505) 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 56: *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 56: *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 477.
- 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*.

Part VI

iSF Programming

In this part

This part contains the following chapters:

CHAPTER 21

Developing an iSF Adapter

An iSF adapter is a replaceable component of the iSF server module that enables you to integrate iSF with any third-party enterprise security service. This chapter explains how to develop and configure a custom iSF adapter implementation.

In this chapter

This chapter discusses the following topics:

iSF Security Architecture

Overview

This section introduces the basic components and concepts of the iSF security architecture, as follows:

- Architecture.
- iSF client.
- iSF client SDK.
- Orbix Security Service.
- iSF adapter SDK.
- iSF adapter.
- Example adapters.

Architecture

Figure 68 gives an overview of the Orbix Security Service, showing how it fits into the overall context of a secure system.

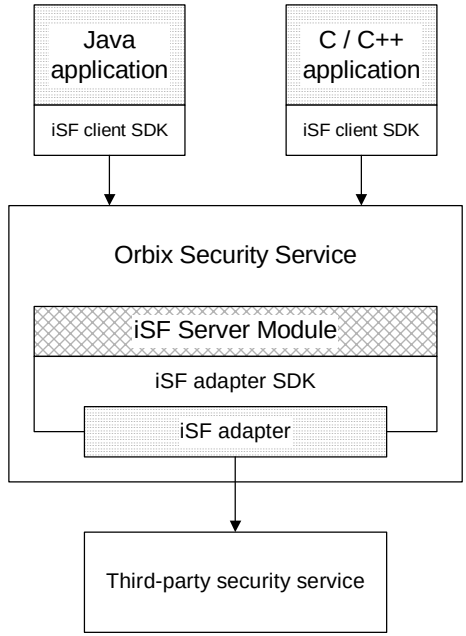

Figure 68: *Overview of the Orbix Security Service*

iSF client

An iSF client is an application that communicates with the Orbix Security Service to perform authentication and authorization operations. The following are possible examples of iSF client applications:

- CORBA servers.
- Artix servers.
- Any server that has a requirement to authenticate its clients.

Hence, an iSF client can also be a server. It is a client only with respect to the Orbix Security Service.

iSF client SDK

The *iSF client SDK* is the programming interface that enables the iSF clients to communicate (usually remotely) with the Orbix Security Service.

Note: The iSF client SDK is only used internally. It is currently not available as a public programming interface.

Orbix Security Service

The Orbix Security Service is a standalone process that acts a thin wrapper layer around the iSF server module. On its own, the iSF server module is a Java library which could be accessed only through local calls. By embedding the iSF server module within the Orbix Security Service, however, it becomes possible to access the security service remotely.

iSF server module

The *iSF server module* is a broker that mediates between iSF clients, which request the security service to perform security operations, and a third-party security service, which is the ultimate repository for security data.

The *iSF server module* has the following special features:

- A replaceable iSF adapter component that enables integration with a third-party enterprise security service.
- A single sign-on feature with user session caching.

iSF adapter SDK

The *iSF adapter SDK* is the Java API that enables a developer to create a custom iSF adapter that plugs into the iSF server module.

iSF adapter

An *iSF adapter* is a replaceable component of the iSF server module that enables you to integrate with any third-party enterprise security service. An iSF adapter implementation provides access to a repository of authentication data and (optionally) authorization data as well.

Example adapters

The following standard adapters are provided with Orbix:

- Lightweight Directory Access Protocol (LDAP).
- File—a simple adapter implementation that stores authentication and authorization data in a flat file.

WARNING: The file adapter is intended for demonstration purposes only. It is not industrial strength and is *not* meant to be used in a production environment.

iSF Server Module Deployment Options

Overview

The iSF server module, which is fundamentally implemented as a Java library, can be deployed in one of the following ways:

- CORBA service.

CORBA service

The iSF server module can be deployed as a CORBA service (Orbix Security Service), as shown in Figure 69. This is the default deployment model for the iSF server module in Orbix. This deployment option has the advantage that any number of distributed iSF clients can communicate with the iSF server module over IIOP/TLS.

With this type of deployment, the iSF server module is packaged as an application plug-in to the Orbix *generic server* (just like any of the other standard Orbix services). The Orbix Security Service can be launched by the `itsecurity` executable and basic configuration is set in the `iona_services.security` scope of the Orbix configuration file.

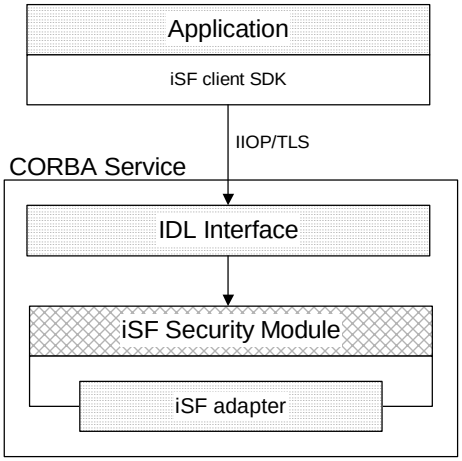

Figure 69: *iSF Server Module Deployed as a CORBA Service*

iSF Adapter Overview

Overview

This section provides an overview of the iSF adapter architecture. The modularity of the iSF server module design makes it relatively straightforward to implement a custom iSF adapter written in Java.

Standard iSF adapters

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

- File adapter.
- LDAP adapter.

Custom iSF adapters

The iSF server module architecture also allows you to implement your own custom iSF adapter and use it instead of a standard adapter.

Main elements of a custom iSF adapter

The main elements of a custom iSF adapter are, as follows:

- Implementation of the ISF Adapter Java interface.
- Configuration of the ISF adapter using the iSF properties file.

Implementation of the ISF Adapter Java interface

The only code that needs to be written to implement an iSF adapter is a class to implement the `IS2Adapter` Java interface. The adapter implementation class should respond to authentication requests either by checking a repository of user data or by forwarding the requests to a third-party enterprise security service.

Configuration of the ISF adapter using the iSF properties file

The iSF adapter is configured by setting Java properties in the `is2.properties` file. The `is2.properties` file stores two kinds of configuration data for the iSF adapter:

- Configuration of the iSF server module to load the adapter—see "Configuring iSF to Load the Adapter" on page 538.
- Configuration of the adapter itself—see "Setting the Adapter Properties" on page 539.

Implementing the IS2Adapter Interface

Overview

The `com.iona.security.is2adapter` package defines an `IS2Adapter` Java interface, which a developer must implement to create a custom iSF adapter. The methods defined on the `ISFAdapter` class are called by the iSF server module in response to requests received from iSF clients.

This section describes a simple example implementation of the `IS2Adapter` interface, which is capable of authenticating a single test user with hard-coded authorization properties.

Test user

The example adapter implementation described here permits authentication of just a single user, `test_user`. The test user has the following authentication data:

```
Username: test_user
Password: test_password
```

and the following authorization data:

- The user's global realm contains the `GuestRole` role.
- The user's `EngRealm` realm contains the `EngineerRole` role.
- The user's `FinanceRealm` realm contains the `AccountantRole` role.

iSF adapter example

Example 57 shows a sample implementation of an iSF adapter class, `ExampleAdapter`, that permits authentication of a single user. The user's username, password, and authorization are hard-coded. In a realistic system, however, the user data would probably be retrieved from a database or from a third-party enterprise security system.

Example 57: *Sample ISF Adapter Implementation*

```
import com.iona.security.azmgr.AuthorizationManager;
import com.iona.security.common.AuthenticatedPrincipal;
import com.iona.security.common.Realm;
import com.iona.security.common.Role;
import com.iona.security.is2adapter.IS2Adapter;
import com.iona.security.is2adapter.IS2AdapterException;
import java.util.Properties;
import java.util.ArrayList;
import java.security.cert.X509Certificate;
```

Example 57: *Sample ISF Adapter Implementation*

```
import org.apache.log4j.*;
import java.util.ResourceBundle;

import java.util.MissingResourceException;

public class ExampleAdapter implements IS2Adapter {

    public final static String EXAMPLE_PROPERTY =
   "example_property";

    public final static String ADAPTER_NAME = "ExampleAdapter";

1   private final static String MSG_EXAMPLE_ADAPTER_INITIALIZED
    = "initialized";
   private final static String MSG_EXAMPLE_ADAPTER_CLOSED
    = "closed";
  private final static String MSG_EXAMPLE_ADAPTER_AUTHENTICATE
    = "authenticate";
     private final static String
    MSG_EXAMPLE_ADAPTER_AUTHENTICATE_REALM  =
    "authenticate_realm";
     private final static String
    MSG_EXAMPLE_ADAPTER_AUTHENTICATE_OK     = "authenticateok";
  private final static String MSG_EXAMPLE_ADAPTER_GETAUTHINFO
    = "getauthinfo";
     private final static String
    MSG_EXAMPLE_ADAPTER_GETAUTHINFO_OK      = "getauthinfook";

    private ResourceBundle _res_bundle = null;

2    private static Logger LOG =
    Logger.getLogger(ExampleAdapter.class.getName());

    public ExampleAdapter() {
3    _res_bundle = ResourceBundle.getBundle("ExampleAdapter");
    LOG.setResourceBundle(_res_bundle);
    }


4    public void initialize(Properties props)
            throws IS2AdapterException {

        LOG.l7dlog(Priority.INFO, ADAPTER_NAME + "." +
   MSG_EXAMPLE_ADAPTER_INITIALIZED,null);
```

Example 57: *Sample ISF Adapter Implementation*

```
        // example property
        String propVal = props.getProperty(EXAMPLE_PROPERTY);
        LOG.info(propVal);

    }

5   public void close() throws IS2AdapterException {
       LOG.l7dlog(Priority.INFO, ADAPTER_NAME + "." +
MSG_EXAMPLE_ADAPTER_CLOSED, null);
    }


6   public AuthenticatedPrincipal authenticate(String username,
String password)
    throws IS2AdapterException {

7       LOG.l7dlog(Priority.INFO, ADAPTER_NAME + "." +
MSG_EXAMPLE_ADAPTER_AUTHENTICATE,new
Object[]{username,password},null);

        AuthenticatedPrincipal ap = null;
        try{
            if (username.equals("test_user")
             && password.equals("test_password")){
8               ap = getAuthorizationInfo(new
AuthenticatedPrincipal(username));
            }
            else {
                LOG.l7dlog(Priority.WARN, ADAPTER_NAME + "." +
IS2AdapterException.WRONG_NAME_PASSWORD,null);
9               throw new IS2AdapterException(_res_bundle,this,
IS2AdapterException.WRONG_NAME_PASSWORD, new
Object[]{username});
            }

        } catch (Exception e) {
            LOG.l7dlog(Priority.WARN, ADAPTER_NAME + "." +
IS2AdapterException.AUTH_FAILED,e);
            throw new IS2AdapterException(_res_bundle,this,
IS2AdapterException.AUTH_FAILED, new Object[]{username}, e);
        }

        LOG.l7dlog(Priority.WARN, ADAPTER_NAME + "." +
MSG_EXAMPLE_ADAPTER_AUTHENTICATE_OK,null);
        return ap;
```

Example 57: *Sample ISF Adapter Implementation*

```
    }

10  public AuthenticatedPrincipal authenticate(String realmname,
String username, String password)
    throws IS2AdapterException {

        LOG.l7dlog(Priority.INFO, ADAPTER_NAME + "." +
MSG_EXAMPLE_ADAPTER_AUTHENTICATE_REALM,new
Object[]{realmname,username,password},null);

        AuthenticatedPrincipal ap = null;
        try{
            if (username.equals("test_user")
             && password.equals("test_password")){
11              AuthenticatedPrincipal principal = new
AuthenticatedPrincipal(username);
                principal.setCurrentRealm(realmname);
                ap = getAuthorizationInfo(principal);
            }
            else {
                LOG.l7dlog(Priority.WARN, ADAPTER_NAME + "." +
IS2AdapterException.WRONG_NAME_PASSWORD,null);
               throw new IS2AdapterException(_res_bundle, this,
IS2AdapterException.WRONG_NAME_PASSWORD, new
Object[]{username});
            }

        } catch (Exception e) {
            LOG.l7dlog(Priority.WARN, ADAPTER_NAME + "." +
IS2AdapterException.AUTH_FAILED,e);
            throw new IS2AdapterException(_res_bundle, this,
IS2AdapterException.AUTH_FAILED, new Object[]{username}, e);
        }

        LOG.l7dlog(Priority.WARN, ADAPTER_NAME + "." +
MSG_EXAMPLE_ADAPTER_AUTHENTICATE_OK,null);
        return ap;
    }

12  public AuthenticatedPrincipal authenticate(X509Certificate
certificate)
    throws IS2AdapterException {
            throw new IS2AdapterException(
                _res_bundle, this,
IS2AdapterException.NOT_IMPLEMENTED
```

Example 57: *Sample ISF Adapter Implementation*

```
            );
    }

13  public AuthenticatedPrincipal authenticate(String realm,
X509Certificate certificate)
    throws IS2AdapterException {
            throw new IS2AdapterException(
                _res_bundle, this,
IS2AdapterException.NOT_IMPLEMENTED
            );
    }


14  public AuthenticatedPrincipal
getAuthorizationInfo(AuthenticatedPrincipal principal) throws
IS2AdapterException{

        LOG.l7dlog(Priority.INFO, ADAPTER_NAME + "." +
MSG_EXAMPLE_ADAPTER_GETAUTHINFO,new
Object[]{principal.getUserID()},null);

        AuthenticatedPrincipal ap = null;
        String username = principal.getUserID();
        String realmname = principal.getCurrentRealm();

        try{
            if (username.equals("test_user")) {
15              ap = new AuthenticatedPrincipal(username);
16              ap.addRole(new Role("GuestRole", ""));

17              if (realmname == null || (realmname != null &&
realmname.equals("EngRealm")))
                {
                    ap.addRealm(new Realm("EngRealm", ""));
                    ap.addRole("EngRealm", new
Role("EngineerRole", ""));
                }
18              if (realmname == null || (realmname != null &&
realmname.equals("FinanceRealm")))
                {
                    ap.addRealm(new Realm("FinanceRealm",""));
                    ap.addRole("FinanceRealm", new
Role("AccountantRole", ""));
                }
            }
```

Example 57: *Sample ISF Adapter Implementation*

```
            else {
                LOG.l7dlog(Priority.WARN, ADAPTER_NAME + "." +
IS2AdapterException.USER_NOT_EXIST, new Object[]{username},
null);
                throw new IS2AdapterException(_res_bundle, this,
IS2AdapterException.USER_NOT_EXIST, new Object[]{username});
            }

        } catch (Exception e) {
            LOG.l7dlog(Priority.WARN, ADAPTER_NAME + "." +
IS2AdapterException.AUTH_FAILED,e);
            throw new IS2AdapterException(_res_bundle, this,
IS2AdapterException.AUTH_FAILED, new Object[]{username}, e);
        }

        LOG.l7dlog(Priority.WARN, ADAPTER_NAME + "." +
MSG_EXAMPLE_ADAPTER_GETAUTHINFO_OK,null);
        return ap;
    }


19  public AuthenticatedPrincipal getAuthorizationInfo(String
username) throws IS2AdapterException{

            // this method has been deprecated
            throw new IS2AdapterException(
                _res_bundle, this,
IS2AdapterException.NOT_IMPLEMENTED
            );
    }

20  public AuthenticatedPrincipal getAuthorizationInfo(String
realmname, String username) throws IS2AdapterException{

            // this method has been deprecated
            throw new IS2AdapterException(
                _res_bundle, this,
IS2AdapterException.NOT_IMPLEMENTED
            );
    }


21  public ArrayList getAllUsers()
    throws IS2AdapterException {
```

Example 57: *Sample ISF Adapter Implementation*

```
            throw new IS2AdapterException(
                _res_bundle, this,
   IS2AdapterException.NOT_IMPLEMENTED
            );

    }



22      public void logout(AuthenticatedPrincipal ap) throws
    IS2AdapterException {

    }
}
```

The preceding iSF adapter code can be explained as follows:

1. These lines list the keys to the messages from the adapter's resource bundle. The resource bundle stores messages used by the Log4J logger and exceptions thrown in the adapter.
2. This line creates a Log4J logger.
3. This line loads the resource bundle for the adapter.
4. The `initialize()` method is called just after the adapter is loaded. The properties passed to the `initialize()` method, `props`, are the adapter properties that the iSF server module has read from the `is2.properties` file.

   See "Setting the Adapter Properties" on page 539 for more details.
5. The `close()` method is called to shut down the adapter. This gives you an opportunity to clean up and free resources used by the adapter.
6. This variant of the `IS2Adapter.authenticate()` method is called whenever an iSF client calls `AuthManager.authenticate()` with username and password parameters.

   In this simple demonstration implementation, the `authenticate()` method recognizes only one user, `test_user`, with password, `test_password`.
7. This line calls a Log4J method in order to log a localized and parametrized message to indicate that the authenticate method has been called with the specified username and password values. Since

all the keys in the resource bundle begin with the adapter name, the adapter name is prepended to the key. The `l7dlog()` method is used because it automatically searches the resource beundle which was set previously by the loggers `setResourceBundle()` method.

8. If authentication is successful; that is, if the name and password passed in match `test_user` and `test_password`, the `getAuthorizationInfo()` method is called to obtain an `AuthenticatedPrincipal` object populated with *all* of the user's realms and role
9. If authentication fails, an `IS2AdapterException` is raised with minor code `IS2AdapterException.WRONG_NAME_PASSWORD`.
   The resource bundle is passed to the exception as it accesses the exception message from the bundle using the key, `ExampleAdapter.wrongUsernamePassword`.
10. This variant of the `IS2Adapter.authenticate()` method is called whenever an iSF client calls `AuthManager.authenticate()` with realm name, username and password parameters.
    This method differs from the preceding username/password `authenticate()` method in that only the authorization data for the specified realm and the global realm are included in the return value.
11. If authentication is successful, the `getAuthorizationInfo()` method is called to obtain an `AuthenticatedPrincipal` object populated with the authorization data from the specified realm and the global realm.
12. This variant of the `IS2Adapter.authenticate()` method is called whenever an iSF client calls `AuthManager.authenticate()` with an X.509 certificate parameter.
13. This variant of the `IS2Adapter.authenticate()` method is called whenever an iSF client calls `AuthManager.authenticate()` with a realm name and an X.509 certificate parameter.
    This method differs from the preceding certificate `authenticate()` method in that only the authorization data for the specified realm and the global realm are included in the return value.
14. This method should create an `AuthenticatedPrincipal` object for the `username` user. If a realm is *not* specified in the principal, the `AuthenticatedPrincipal` is populated with all realms and roles for this

user. If a realm *is* specified in the principal, the `AuthenticatedPrincipal` is populated with authorization data from the specified realm and the global realm only.

15. This line creates a new `AuthenticatedPrincipal` object for the `username` user to hold the user's authorization data.
16. This line adds a `GuestRole` role to the global realm, `IONAGlobalRealm`, using the single-argument form of `addRole()`. Roles added to the global realm implicitly belong to every named realm as well.
17. This line checks if no realm is specified in the principal or if the realm, `EngRealm`, is specified. If either of these is true, the following lines add the authorization realm, `EngRealm`, to the `AuthenticatedPrincipal` object and add the `EngineerRole` role to the `EngRealm` authorization realm.
18. This line checks if no realm is specified in the principal or if the realm, `FinanceRealm`, is specified. If either of these is true, the following lines add the authorization realm, `FinanceRealm`, to the `AuthenticatedPrincipal` object and add the `AccountantRole` role to the `FinanceRealm` authorization realm.
19. Since SSO was introduced to Orbix, this variant of the `IS2Adapter.getAuthorizationInfo()` method has been deprecated. The method `IS2Adapter.getAuthorizationInfo(AuthenticatedPrincipal principal)` should be used instead
20. Since SSO was introduced to Orbix, this variant of the `IS2Adapter.getAuthorizationInfo()` method has also been deprecated. The method `IS2Adapter.getAuthorizationInfo(AuthenticatedPrincipal principal)` should be used instead
21. The `getAllUsers()` method is currently not used by the iSF server module during runtime. Hence, there is no need to implement this method currently.

22. When the `logout()` method is called, you can perform cleanup and release any resources associated with the specified user principal. The iSF server module calls back on `IS2Adapter.logout()` either in response to a user calling `AuthManager.logout()` explicitly or after an SSO session has timed out.

Deploying the Adapter

Overview

This section explains how to deploy a custom iSF adapter.

In this section

This section contains the following subsections:

Configuring iSF to Load the Adapter

Overview

You can configure the iSF server module to load a custom adapter by setting the following properties in the iSF server module's `is2.properties` file:

- Adapter name.
- Adapter class.

Adapter name

The iSF server module loads the adapter identified by the `com.iona.isp.adapters` property. Hence, to load a custom adapter, *`AdapterName`*, set the property as follows:

```
com.iona.isp.adapters=AdapterName
```

Note: In the current implementation, the iSF server module can load only a single adapter at a time.

Adapter class

The name of the adapter class to be loaded is specified by the following property setting:

```
com.iona.isp.adapter.AdapterName.class=AdapterClass
```

Example adapter

For example, the example adapter provided shown previously can be configured to load by setting the following properties:

```
com.iona.isp.adapters=example
com.iona.isp.adapter.example.class=isfadapter.ExampleAdapter
```

Setting the Adapter Properties

Overview

This subsection explains how you can set properties for a specific custom adapter in the `is2.properties` file.

Adapter property name format

All configurable properties for a custom file adapter, *`AdapterName`*, should have the following format:

`com.iona.isp.adapter.`*`AdapterName`*`.param.`*`PropertyName`*

Truncation of property names

Adapter property names are truncated before being passed to the iSF adapter. That is, the `com.iona.ispadapter.`*`AdapterName`*`.param` prefix is stripped from each property name.

Example

For example, given an adapter named `ExampleAdapter` which has two properties, `host` and `port`, these properties would be set as follows in the `is2.properties` file:

```
com.iona.isp.adapter.example.param.example_property="This is an
   example property"
```

Before these properties are passed to the iSF adapter, the property names are truncated as if they had been set as follows:

```
example_property="This is an example property"
```

Accessing properties from within an iSF adapter

The adapter properties are passed to the iSF adapter through the `com.iona.security.is2adapter.IS2Adapter.initialize()` callback method. For example:

```
...
public void initialize(java.util.Properties props)
throws IS2AdapterException {
    // Access a property through its truncated name.
    String propVal = props.getProperty("PropertyName")
    ...
}
```

Loading the Adapter Class and Associated Resource Files

Overview

You need to make appropriate modifications to your CLASSPATH to ensure that the iSF server module can find your custom adapter class.

In all cases, the location of the file used to configure Log4j logging can be set using the `log4j.configuration` property in the `is2.properties` file.

CORBA service

By default, the Orbix Security Service uses the `iona_services.security` scope in your Orbix configuration file (or configuration repository service). Modify the `plugins:java_server:classpath` variable to include the directory containing the compiled adapter class and the adapter's resource bundle. The `plugins:java_server:classpath` variable uses the value of the SECURITY_CLASSPATH variable.

For example, if the adapter class and adapter resource bundle are located in the *OrbixInstallDir*\ExampleAdapter directory, you should set the SECURITY_CLASSPATH variable as follows:

```
# Orbix configuration file
SECURITY_CLASSPATH =
   "OrbixInstallDir\ExampleAdapter;OrbixInstallDir\etc\domains;O
   rbixInstallDir\etc\domains\DomainName\;OrbixInstallDir\asp\V
   ersion\lib\security.jar";
```

The Orbix Security Service launches a Java process which uses the classpath defined in the `securityserver_ce.xml` file which is located in the *OrbixInstallDir*/etc/domains/*DomainName*/resources directory. This classpath also needs to be modified.

In this case, you must also modify the `ce:loader` element of `securityserver_ce.xml` file, as shown in the following example:

```
# securityserver_ce.xml file
...
     <ce:loader>
      <ce:location>OrbixInstallDir\ExampleAdapter</ce:location>
      <ce:location>${java.home}/../lib/tools.jar</ce:location>

      <ce:location>OrbixInstallDir\etc\domains</ce:location>

   <ce:location>OrbixInstallDir\asp\Version\bin\..\lib\security.
   jar</ce:location>
   </ce:loader>
...
```

APPENDIX A

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 Schannel SSL/TLS toolkit, you would set configuration variables as follows:

```
initial_references:IT_TLS_Toolkit:plugin = "schannel_toolkit";
plugins:schannel_toolkit:shlib_name = "it_tls_schannel";
```

plugins:atli2_tls

The `plugins:atli2_tls` namespace contains the following variables:

- cert_store_protocol
- cert_store_provider
- kmf_algorithm
- tmf_algorithm
- use_jsse_tk

cert_store_protocol

(Java only) This variable is used in conjunction with `policies:tls:use_external_cert_store` to configure Orbix to use an external certificate store. Orbix passes the value of this variable as the `protocol` argument to the `javax.net.ssl.SSLContext.getInstance()` method. To obtain a list of possible values for this variable, consult the documentation for your third-party JSSE/JCS security provider.

For example, if your application is using the Sun JSSE security provider, you can configure the certificate store to use the SSLv3 protocol as follows:

```
plugins:atli2_tls:cert_store_protocol = "SSLv3";
```

cert_store_provider

(Java only) This variable is used in conjunction with `policies:tls:use_external_cert_store` to configure Orbix to use an external certificate store. Orbix passes the value of this variable as the `provider` argument to the `javax.net.ssl.SSLContext.getInstance()` method. To obtain a list of possible values for this variable, consult the documentation for your third-party JSSE/JCS security provider.

For example, if your application is using the Sun JSSE security provider, you can configure the certificate store provider as follows:

```
plugins:atli2_tls:cert_store_provider = "SunJSSE";
```

kmf_algorithm

(Java only) This variable is used in conjunction with `policies:tls:use_external_cert_store` to configure Orbix to use an external certificate store. Orbix passes the value of this variable as the `algorithm` argument to the `javax.net.ssl.KeyManagerFactory.getInstance()` method, overriding the value of the `ssl.KeyManagerFactory.algorithm` property set in the `java.security` file. To obtain a list of possible values for this variable, consult the documentation for your third-party JSSE/JCS security provider.

For example, if your application is using the Sun JSSE security provider, you can configure the key manager factory to use the following algorithm:

```
plugins:atli2_tls:kmf_algorithm = "SunX509";
```

tmf_algorithm

(Java only) This variable is used in conjunction with `policies:tls:use_external_cert_store` to configure Orbix to use an external certificate store. Orbix passes the value of this variable as the `algorithm` argument to the `javax.net.ssl.TrustManagerFactory.getInstance()` method, overriding the value of the `ssl.TrustManagerFactory.algorithm` property set in the `java.security` file. To obtain a list of possible values for this variable, consult the documentation for your third-party JSSE/JCS security provider.

For example, if your application is using the Sun JSSE security provider, you can configure the trust manager factory to use the following algorithm:

```
plugins:atli2_tls:tmf_algorithm = "SunX509";
```

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

- `allow_csi_reply_without_service_context`.
- `ClassName`.
- `shlib_name`.
- `use_legacy_policies`.

allow_csi_reply_without_service_context

(Java only) Boolean variable that specifies whether a CSIv2 client enforces strict checking for the presence of a CSIv2 service context in the reply it receives from the server.

Up until Orbix 6.2 SP1, the Java implementation of the CSIv2 protocol permitted replies from a CSIv2 enabled server even if the server did not send a CSIv2 response. From Orbix 6.2 SP1 onwards, this variable determines whether or not the client checks for a CSIv2 response.

If the variable is set to `false`, the client enforces strict checking on the server reply. If there is no CSIv2 service context in the reply, a `NO_PERMISSION` exception with the minor code, `BAD_SAS_SERVICE_CONTEXT`, is thrown by the client.

If the variable is set to `true`, the client does *not* enforce strict checking on the reply. If there is no CSIv2 service context in the reply, the client does not raise an exception.

Default is `true`.

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.

use_legacy_policies

Boolean variable that specifies whether the application can be programmed using the new CSIv2 policy types or the older (legacy) CSIv2 policy types.

If `plugins:csi:use_legacy_policies` is set to `true`, you can program CSIv2 using the following policies:

- `IT_CSI::AuthenticationServicePolicy`
- `IT_CSI::AttributeServicePolicy`

If `plugins:csi:use_legacy_policies` is set to `false`, you can program CSIv2 using the following policies:

- `IT_CSI::AttributeServiceProtocolClient`
- `IT_CSI::AttributeServiceProtocolServer`

Default is `false`.

plugins:gsp

The `plugins:gsp` namespace includes variables that specify settings for the Generic Security Plugin (GSP). This provides authorization by checking a user's roles against the permissions stored in an action-role mapping file. It includes the following:

- `accept_asserted_authorization_info`
- `action_role_mapping_file`
- `assert_authorization_info`
- `authentication_cache_size`
- `authentication_cache_timeout`
- `authorization_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`
- `retrieve_isf_auth_principal_info_for_all_realms`
- `sso_server_certificate_constraints`
- `use_client_load_balancing`

accept_asserted_authorization_info

If `false`, SAML data is not read from incoming connections. Default is `true`.

action_role_mapping_file

Specifies the action-role mapping file URL. For example:

```
plugins:gsp:action_role_mapping_file =
   "file:///my/action/role/mapping";
```

assert_authorization_info

If `false`, SAML data is not sent on outgoing connections. Default is `true`.

authentication_cache_size

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

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

authentication_cache_timeout

The time (in seconds) after which a credential is considered *stale*. Stale credentials are removed from the cache and the server must re-authenticate with the 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 setting that enables X.509 certificate-based authentication on the server side using the Orbix security service.

Default is `false`.

retrieve_isf_auth_principal_info_for_all_realms

A boolean setting that determines whether the GSP plug-in retrieves role and realm data for all realms, when authenticating user credentials. If `true`, the GSP plug-in retrieves the user's role and realm data for all realms; if `false`, the GSP plug-in retrieves the user's role and realm data only for the realm specified by `plugins:gsp:authorization_realm`.

Setting this variable to `false` can provide a useful performance optimization in some applications. But you must take special care to configure the application correctly for making operation invocations between different realms.

Default is `true`.

sso_server_certificate_constraints

A special certificate constraints policy that applies *only* to the SSL/TLS connection between the client and the SSO login server. For details of the pattern constraint language, see "Applying Constraints to Certificates" on page 545.

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

(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.

cert_expiration_warning_days

(Since Orbix 6.2 SP1) Specifies the threshold for the number of days left to certificate expiration, before Orbix issues a warning. If the application's own certificate is due to expire in less than the specified number of days, Orbix issues a warning message to the log.

Default is 31 days.

See also the following related configuration variables:

`plugins:iiop_tls:enable_warning_for_approaching_cert_expiration`
`plugins:iiop_tls:own_credentials_warning_cert_constraints`

delay_credential_gathering_until_handshake

(Windows and Schannel only) This client configuration variable provides an alternative to using the `principal_sponsor` variables to specify an application's own certificate. When this variable is set to `true` and `principal_sponsor:use_principal_sponsor` is set to `false`, the client delays sending its certificate to a server. The client will wait until the server *explicitly* requests the client to send its credentials during the SSL/TLS handshake.

This configuration variable can be used in conjunction with the `plugins:schannel:prompt_with_credential_choice` configuration variable.

enable_iiop_1_0_client_support

This variable enables client-side interoperability of 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.

enable_warning_for_approaching_cert_expiration

(Since Orbix 6.2 SP1) Enables warnings to be sent to the log, if an application's own ceritificate is imminently about to expire. The boolean value can have the following values: `true`, enables the warning feature; `false`, disables the warning feature.

Default is `true`.

See also the following related configuration variables:

`plugins:iiop_tls:cert_expiration_warning_days`
`plugins:iiop_tls:own_credentials_warning_cert_constraints`

incoming_connections:hard_limit

Specifies the maximum number of incoming (server-side) connections permitted to IIOP. IIOP does not accept new connections above this limit. Defaults to -1 (disabled).

When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:incoming_connections:hard_limit` variable's value.

Please see the chapter on ACM in the *CORBA Programmer's Guide* for further details.

incoming_connections:soft_limit

Specifies the number of connections at which IIOP should begin closing incoming (server-side) connections. Defaults to -1 (disabled).

When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:incoming_connections:soft_limit` variable's value.

Please see the chapter on ACM in the *CORBA Programmer's Guide* for further details.

outgoing_connections:hard_limit

When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:outgoing_connections:hard_limit` variable's value.

outgoing_connections:soft_limit

When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:outgoing_connections:soft_limit` variable's value.

own_credentials_warning_cert_constraints

(Since Orbix 6.2 SP1) Set this certificate constraints variable, if you would like to avoid deploying certain certificates as an own certificate. A warning is issued, if the own certificate's subject DN matches the constraints specified by this variable (see "Applying Constraints to Certificates" on page 545 for details of the constraint language). For example, you might want to generate a warning in case you accidentally deployed an IONA demonstration certificate.

Default is an empty list, `[]`.

Note: This warning is *not* related to certificate expiration and works independently of the certificate expiration warning.

tcp_listener:reincarnate_attempts

(Windows only)

`plugins: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:kdm

The `plugins:kdm` namespace contains the following variables:

- cert_constraints
- iiop_tls:port
- checksums_optional

cert_constraints

Specifies the list of certificate constraints for principals attempting to open a connection to the KDM server plug-in. See "Applying Constraints to Certificates" on page 545 for a description of the certificate constraint syntax.

To protect the sensitive data stored within it, the KDM applies restrictions on which entities are allowed talk to it. A security administrator should choose certificate constraints that restrict access to the following principals:

- The locator service (requires read-only access).
- The `kdm_adm` plug-in, which is normally loaded into the itadmin utility (requires read-write access).

All other principals should be blocked from access. For example, you might define certificate constraints similar to the following:

```
plugins:kdm:cert_constraints =
   ["C=US,ST=Massachusetts,O=ABigBank*,CN=Secure admin*",
   "C=US,ST=Boston,O=ABigBank*,CN=Orbix2000 Locator Service*"]
```

Your choice of certificate constraints will depend on the naming scheme for your subject names.

iiop_tls:port

Specifies the well known IP port on which the KDM server listens for incoming calls.

checksums_optional

When equal to `false`, the secure information associated with a server must include a checksum; when equal to `true`, the presence of a checksum is optional. Default is `false`.

plugins:kdm_adm

The `plugins:kdm_adm` namespace contains the following variable:

- cert_constraints

cert_constraints

Specifies the list of certificate constraints that are applied when the KDM administration plug-in authenticates the KDM server. See "Applying Constraints to Certificates" on page 545 for a description of the certificate constraint syntax.

The KDM administration plug-in requires protection against attack from applications that try to impersonate the KDM server. A security administrator should, therefore, choose certificate constraints that restrict access to trusted KDM servers only. For example, you might define certificate constraints similar to the following:

```
plugins:kdm_adm:cert_constraints =
   ["C=US,ST=Massachusetts,O=ABigBank*,CN=IT_KDM*"];
```

Your choice of certificate constraints will depend on the naming scheme for your subject names.

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:schannel

The `plugins:schannel` namespace contains the following variable:

- prompt_with_credential_choice

prompt_with_credential_choice

(Windows and Schannel only) Setting both this variable and the `plugins:iiop_tls:delay_credential_gathering_until_handshake` variable to `true` on the client side allows the user to choose which credentials to use for the server connection. The choice of credentials offered to the user is based on the trusted CAs sent to the client in an SSL/TLS handshake message.

If `prompt_with_credential_choice` is set to `false`, runtime chooses the first certificate it finds in the certificate store that meets the applicable constraints.

The certificate prompt can be replaced by implementing an IDL interface and registering it with the ORB.

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`.

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
- session_caching_policy
- 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 26 can be specified in this list.

Table 26: *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"];
```

session_caching_policy

session_caching_policy specifies whether an ORB caches the session information for secure associations when acting in a client role, a server role, or both. The purpose of session caching is to enable closed connections to be re-established quickly. The following values are supported:

CACHE_NONE(default)

CACHE_CLIENT
CACHE_SERVER
CACHE_SERVER_AND_CLIENT

The policy can also be set programmatically using the IT_TLS_API::SessionCachingPolicy CORBA policy.

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.

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.

Note: In Orbix 6.1 SP1 and Orbix 6.2, the `policies:https` configuration variables are available *only* in the Java implementation of the `https` plug-in.

The `policies:https` namespace 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

(Java only) 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

(Java only) A list of constraints applied to peer certificates—see "Applying Constraints to Certificates" on page 545 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

(Java only) 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

(Java only) 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

(Java only) 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

(Java only) 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

(Java only) 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 27: *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

Table 27: *Mechanism Policy Cipher Suites*

Null Encryption, Integrity and Authentication Ciphers	Standard Ciphers
	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

(Java only) 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

(Java only) 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

(Java only) 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

(Java only) 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

(Java only) 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`
- `session_caching_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 28: *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 28: *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`.

session_caching_policy

This policy overides `policies:session_caching_policy` for the `iiop_tls` plugin.

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.

policies:tls

The following variables are in this namespace:

- use_external_cert_store

use_external_cert_store

(Java only) A binary variable that configures Orbix to check for the presence of a third-party certificate store. The possible values are: `true`, to check for the presence of an external certificate store, and `false`, to use the built-in certificate store (that is, certificate location specified by the principal sponsor).

The default is `false`.

This variable has no effect unless you also configure your Java application to use an external security provider—see the description of the `plugins:atli2_tls:use_jsse_tk` configuration variable for more details.

This policy variable must be used in conjunction with the following configuration variables:

```
plugins:atli2_tls:cert_store_provider
plugins:atli2_tls:cert_store_protocol
```

You can also optionally set the following configuration variables (which override the corresponding properties in the `java.security` file):

```
plugins:atli2_tls:kmf_algorithm
plugins:atli2_tls:tmf_algorithm
```

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.
`pkcs11`	Java only. The authentication data is provided by a smart card.
`security_label`	Windows and Schannel only. The authentication data is specified by supplying the common name (CN) from an application certificate's subject DN.

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 `pkcs11` (smart card) authentication method, the following authentication data can be provided in `auth_method_data`:

`provider`	A name that identifies the underlying PKCS #11 toolkit used by Orbix to communicate with the smart card. The toolkit currently used by Orbix has the provider name `dkck132.dll` (from Baltimore).
`slot`	The number of a particular slot on the smart card (for example, `0`) containing the user's credentials.
`pin`	A PIN to gain access to the smart card—*optional*. It is bad practice to supply the PIN from configuration for deployed systems. If the PIN is not supplied, the user is prompted for it.

For the `security_label` authentication method on Windows, the following authentication data can be provided in `auth_method_data`:

`label`	(Windows and Schannel only.) The common name (CN) from an application certificate's subject DN

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";
```

principal_sponsor:https

The `principal_sponsor:https` namespace provides configuration variables that enable you to specify the *own credentials* used with the HTTPS transport. The variables in the `principal_sponsor:https` namespace (which are specific to the HTTPS protocol) have precedence over the analogous variables in the `principal_sponsor` namespace.

Note: In Orbix 6.1 SP1 and Orbix 6.2, the `principal_sponsor:https` configuration variables are available only in the Java implementation of the `https` plug-in.

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

(Java only) `use_principal_sponsor` specifies whether an attempt is made to obtain credentials automatically. Defaults to `false`. If set to `true`, the following `principal_sponsor:https` variables must contain data in order for anything to actually happen:

- `auth_method_id`
- `auth_method_data`

auth_method_id

(Java only) `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

For example, you can select the `pkcs12_file` authentication method as follows:

```
principal_sponsor:auth_method_id = "pkcs12_file";
```

auth_method_data

(Java only) `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 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"];
```

APPENDIX B

iSF Configuration

This appendix provides details of how to configure the Orbix security server.

In this appendix

This appendix contains the following sections:

Properties File Syntax

Overview

The Orbix security service uses standard Java property files for its configuration. Some aspects of the Java properties file syntax are summarized here for your convenience.

Property definitions

A property is defined with the following syntax:

```
<PropertyName>=<PropertyValue>
```

The *<PropertyName>* is a compound identifier, with each component delimited by the . (period) character. For example, `is2.current.server.id`. The *<PropertyValue>* is an arbitrary string, including all of the characters up to the end of the line (embedded spaces are allowed).

Specifying full pathnames

When setting a property equal to a filename, you normally specify a full pathname, as follows:

UNIX

```
/home/data/securityInfo.xml
```

Windows

```
D:/iona/securityInfo.xml
```

or, if using the backslash as a delimiter, it must be escaped as follows:

```
D:\\iona\\securityInfo.xml
```

Specifying relative pathnames

If you specify a relative pathname when setting a property, the root directory for this path must be added to the Orbix security service's classpath. For example, if you specify a relative pathname as follows:

UNIX

```
securityInfo.xml
```

The security service's classpath must include the file's parent directory:

```
CLASSPATH = /home/data/:<rest_of_classpath>
```

iSF Properties File

Overview

An iSF properties file is used to store the properties that configure a specific Orbix security service instance. Generally, every Orbix security service instance should have its own iSF properties file. This section provides descriptions of all the properties that can be specified in an iSF properties file.

File location

The default location of the iSF properties file is the following:

```
OrbixInstallDir/etc/domains/DomainName/server_Host/is2.properties
```

In general, the iSF properties file location is specified in the Orbix configuration by setting the `is2.properties` property in the `plugins:java_server:system_properties` property list.

For example, on UNIX the security server's property list is normally initialized in the `iona_services.security` configuration scope as follows:

```
# Orbix configuration file
...
iona_services {
    ...
    security {
        ...
        plugins:java_server:system_properties =
   ["org.omg.CORBA.ORBClass=com.iona.corba.art.artimpl.ORBImpl",
   "org.omg.CORBA.ORBSingletonClass=com.iona.corba.art.artimpl.ORBSingleton",
   "is2.properties=ASPInstallDir/etc/domains/DomainName/is2.properties"];
        ...
    };
};
```

List of properties

The following properties can be specified in the iSF properties file:

check.kdc.running

A boolean property that specifies whether or not the Artix security service should check whether the Kerberos KDC server is running. Default is `false`.

check.kdc.principal

(Used in combination with the `check.kdc.running` property.) Specifies the dummy KDC principal that is used for connecting to the KDC server, in order to check whether it is running or not.

com.iona.isp.adapters

Specifies the iSF adapter type to be loaded by the Orbix security service at runtime. Choosing a particular adapter type is equivalent to choosing an Artix security domain. Currently, you can specify one of the following adapter types:

- `file`
- `LDAP`

For example, you can select the LDAP adapter as follows:

```
com.iona.isp.adapters=LDAP
```

Note: The file adapter is intended for demonstration purposes only. Use of the file adapter is *not* supported in production systems.

com.iona.isp.adapter.file.class

Specifies the Java class that implements the file adapter.

For example, the default implementation of the file adapter provided with Orbix is selected as follows:

```
com.iona.isp.adapter.file.class=com.iona.security.is2adapter.file.FileAuthAdapter
```

com.iona.isp.adapter.file.param.filename

Specifies the name and location of a file that is used by the file adapter to store user authentication data.

For example, you can specify the file, `C:/is2_config/security_info.xml`, as follows:

```
com.iona.isp.adapter.file.param.filename=C:/is2_config/security_info.xml
```

com.iona.isp.adapter.file.params

Obsolete. This property was needed by earlier versions of the Orbix security service, but is now ignored.

com.iona.isp.adapter.LDAP.class

Specifies the Java class that implements the LDAP adapter.

For example, the default implementation of the LDAP adapter provided with Orbix is selected as follows:

```
com.iona.isp.adapter.LDAP.class=com.iona.security.is2adapter.ldap.LdapAdapter
```

com.iona.isp.adapter.LDAP.param.CacheSize

Specifies the maximum LDAP cache size in units of bytes. This maximum applies to the *total* LDAP cache size, including all LDAP connections opened by this Orbix security service instance.

Internally, the Orbix security service uses a third-party toolkit (currently the *iPlanet SDK*) to communicate with an LDAP server. The cache referred to here is one that is maintained by the LDAP third-party toolkit. Data retrieved from the LDAP server is temporarily stored in the cache in order to optimize subsequent queries.

For example, you can specify a cache size of 1000 as follows:

```
com.iona.isp.adapter.LDAP.param.CacheSize=1000
```

com.iona.isp.adapter.LDAP.param.CacheTimeToLive

Specifies the LDAP cache time to-live in units of seconds. For example, you can specify a cache time to-live of one minute as follows:

```
com.iona.isp.adapter.LDAP.param.CacheTimeToLive=60
```

com.iona.isp.adapter.LDAP.param.GroupBaseDN

Specifies the base DN of the tree in the LDAP directory that stores user groups.

For example, you could use the RDN sequence, `DC=iona,DC=com`, as a base DN by setting this property as follows:

```
com.iona.isp.adapter.LDAP.param.GroupBaseDN=dc=iona,dc=com
```

Note: The order of the RDNs is significant. The order should be based on the LDAP schema configuration.

com.iona.isp.adapter.LDAP.param.GroupNameAttr

Specifies the attribute type whose corresponding attribute value gives the name of the user group. The default is `CN`.

For example, you can use the common name, `CN`, attribute type to store the user group's name by setting this property as follows:

```
com.iona.isp.adapter.LDAP.param.GroupNameAttr=cn
```

com.iona.isp.adapter.LDAP.param.GroupObjectClass

Specifies the object class that applies to user group entries in the LDAP directory structure. An object class defines the required and allowed attributes of an entry. The default is `groupOfUniqueNames`.

For example, to specify that all user group entries belong to the `groupOfUniqueNames` object class:

```
com.iona.isp.adapter.LDAP.param.GroupObjectClass=groupofuniquenames
```

com.iona.isp.adapter.LDAP.param.GroupSearchScope

Specifies the group search scope. The search scope is the starting point of a search and the depth from the base DN to which the search should occur. This property can be set to one of the following values:

- `BASE`—Search a single entry (the base object).
- `ONE`—Search all entries immediately below the base DN.
- `SUB`—Search all entries from a whole subtree of entries.

Default is `SUB`.

For example:

```
com.iona.isp.adapter.LDAP.param.GroupSearchScope=SUB
```

com.iona.isp.adapter.LDAP.param.host.*<cluster_index>*

For the *<cluster_index>* LDAP server replica, specifies the IP hostname where the LDAP server is running. The *<cluster_index>* is `1` for the primary server, `2` for the first failover replica, and so on.

For example, you could specify that the primary LDAP server is running on host `10.81.1.100` as follows:

```
com.iona.isp.adapter.LDAP.param.host.1=10.81.1.100
```

com.iona.isp.adapter.LDAP.param.MaxConnectionPoolSize

Specifies the maximum LDAP connection pool size for the Orbix security service (a strictly positive integer). The maximum connection pool size is the maximum number of LDAP connections that would be opened and cached by the Orbix security service. The default is `1`.

For example, to limit the Orbix security service to open a maximum of 50 LDAP connections at a time:

```
com.iona.isp.adapter.LDAP.param.MaxConnectionPoolSize=50
```

com.iona.isp.adapter.LDAP.param.MemberDNAttr

Specifies which LDAP attribute is used to retrieve group members. The LDAP adapter uses the `MemberDNAttr` property to construct a query to find out which groups a user belongs to.

The list of the user's groups is needed to determine the complete set of roles assigned to the user. The LDAP adapter determines the complete set of roles assigned to a user as follows:

1. The adapter retrieves the roles assigned directly to the user.
2. The adapter finds out which groups the user belongs to, and retrieves all the roles assigned to those groups.

Default is `uniqueMember`.

For example, you can select the `uniqueMember` attribute as follows:

```
com.iona.isp.adapter.LDAP.param.MemberDNAttr=uniqueMember
```

com.iona.isp.adapter.LDAP.param.MemberFilter

Specifies how to search for members in a group. The value specified for this property must be an LDAP search filter (can be a custom filter).

com.iona.isp.adapter.LDAP.param.MinConnectionPoolSize

Specifies the minimum LDAP connection pool size for the Orbix security service. The minimum connection pool size specifies the number of LDAP connections that are opened during initialization of the Orbix security service. The default is `1`.

For example, to specify a minimum of 10 LDAP connections at a time:

```
com.iona.isp.adapter.LDAP.param.MinConnectionPoolSize=10
```

com.iona.isp.adapter.LDAP.param.port.*<cluster_index>*

For the *<cluster_index>* LDAP server replica, specifies the IP port where the LDAP server is listening. The *<cluster_index>* is 1 for the primary server, 2 for the first failover replica, and so on. The default is 389.

For example, you could specify that the primary LDAP server is listening on port 636 as follows:

```
com.iona.isp.adapter.LDAP.param.port.1=636
```

com.iona.isp.adapter.LDAP.param.PrincipalUserDN.*<cluster_index>*

For the *<cluster_index>* LDAP server replica, specifies the username that is used to login to the LDAP server (in distinguished name format). This property need only be set if the LDAP server is configured to require username/password authentication.

No default.

com.iona.isp.adapter.LDAP.param.PrincipalUserPassword.*<cluster_index>*

For the *<cluster_index>* LDAP server replica, specifies the password that is used to login to the LDAP server. This property need only be set if the LDAP server is configured to require username/password authentication.

No default.

WARNING: Because the password is stored in plaintext, you must ensure that the `is2.properties` file is readable and writable only by users with administrator privileges.

com.iona.isp.adapter.LDAP.param.RetrieveAuthInfo

Specifies whether or not the Orbix security service retrieves authorization information from the LDAP server. This property selects one of the following alternatives:

- yes—the Orbix security service retrieves authorization information from the LDAP server.
- no—the Orbix security service retrieves authorization information from the iS2 authorization manager..

Default is no.

For example, to use the LDAP server's authorization information:

```
com.iona.isp.adapter.LDAP.param.RetrieveAuthInfo=yes
```

com.iona.isp.adapter.LDAP.param.RoleNameAttr

Specifies the attribute type that the LDAP server uses to store the role name. The default is CN.

For example, you can specify the common name, CN, attribute type as follows:

```
com.iona.isp.adapter.LDAP.param.RoleNameAttr=cn
```

com.iona.isp.adapter.LDAP.param.SSLCACertDir.*<cluster_index>*

For the *<cluster_index>* LDAP server replica, specifies the directory name for trusted CA certificates. All certificate files in this directory are loaded and set as trusted CA certificates, for the purpose of opening an SSL connection to the LDAP server. The CA certificates can either be in DER-encoded X.509 format or in PEM-encoded X.509 format.

No default.

For example, to specify that the primary LDAP server uses the `d:/certs/test` directory to store CA certificates:

```
com.iona.isp.adapter.LDAP.param.SSLCACertDir.1=d:/certs/test
```

com.iona.isp.adapter.LDAP.param.SSLClientCertFile.*<cluster_index>*

Specifies the client certificate file that is used to identify the Orbix security service to the *<cluster_index>* LDAP server replica. This property is needed only if the LDAP server requires SSL/TLS mutual authentication. The certificate must be in PKCS#12 format.

No default.

com.iona.isp.adapter.LDAP.param.SSLClientCertPassword.*<cluster_index>*

Specifies the password for the client certificate that identifies the Orbix security service to the *<cluster_index>* LDAP server replica. This property is needed only if the LDAP server requires SSL/TLS mutual authentication.

WARNING: Because the password is stored in plaintext, you must ensure that the `is2.properties` file is readable and writable only by users with administrator privileges.

com.iona.isp.adapter.LDAP.param.SSLEnabled.*<cluster_index>*

Enables SSL/TLS security for the connection between the Orbix security service and the *<cluster_index>* LDAP server replica. The possible values are yes or no. Default is no.

For example, to enable an SSL/TLS connection to the primary LDAP server:

```
com.iona.isp.adapter.LDAP.param.SSLEnabled.1=yes
```

com.iona.isp.adapter.LDAP.param.UseGroupAsRole

Specifies whether a user's groups should be treated as roles. The following alternatives are available:

- yes—each group name is interpreted as a role name.
- no—for each of the user's groups, retrieve all roles assigned to the group.

This option is useful for some older versions of LDAP, such as iPlanet 4.0, that do not have the role concept.

Default is no.

For example:

```
com.iona.isp.adapter.LDAP.param.UseGroupAsRole=no
```

com.iona.isp.adapter.LDAP.param.UserBaseDN

Specifies the base DN (an ordered sequence of RDNs) of the tree in the LDAP directory that stores user object class instances.

For example, you could use the RDN sequence, DC=iona,DC=com, as a base DN by setting this property as follows:

```
com.iona.isp.adapter.LDAP.param.UserBaseDN=dc=iona,dc=com
```

com.iona.isp.adapter.LDAP.param.UserCertAttrName

Specifies the attribute type that stores a user certificate. The default is userCertificate.

For example, you can explicitly specify the attribute type for storing user certificates to be userCertificate as follows:

```
com.iona.isp.adapter.LDAP.param.UserCertAttrName=userCertificate
```

com.iona.isp.adapter.LDAP.param.UserNameAttr=uid

Specifies the attribute type whose corresponding value uniquely identifies the user. This is the attribute used as the user's login ID. The default is uid.

For example:

```
com.iona.isp.adapter.LDAP.param.UserNameAttr=uid
```

com.iona.isp.adapter.LDAP.param.UserObjectClass

Specifies the attribute type for the object class that stores users. The default is `organizationalPerson`.

For example:

```
com.iona.isp.adapter.LDAP.param.UserObjectClass=organizationalPerson
```

com.iona.isp.adapter.LDAP.param.UserRoleDNAttr

Specifies the attribute type that stores a user's role DN. The default is `nsRoleDn` (from the Netscape LDAP directory schema).

For example:

```
com.iona.isp.adapter.LDAP.param.UserRoleDNAttr=nsroledn
```

com.iona.isp.adapter.LDAP.param.UserSearchFilter

Custom filter for retrieving users. In the current version, `$USER_NAME$` is the only replaceable parameter supported. This parameter would be replaced during runtime by the LDAP adapter with the current User's login ID. This property uses the standard LDAP search filter syntax.

For example:

```
&(uid=$USER_NAME$)(objectclass=organizationalPerson)
```

com.iona.isp.adapter.LDAP.param.UserSearchScope

Specifies the user search scope. This property can be set to one of the following values:

- `BASE`—Search a single entry (the base object).
- `ONE`—Search all entries immediately below the base DN.
- `SUB`—Search all entries from a whole subtree of entries.

Default is `SUB`.

For example:

```
com.iona.isp.adapter.LDAP.param.UserSearchScope=SUB
```

com.iona.isp.adapter.LDAP.param.version

Specifies the LDAP protocol version that the Orbix security service uses to communicate with LDAP servers. The only supported version is 3 (for LDAP v3, http://www.ietf.org/rfc/rfc2251.txt). The default is 3.

For example, to select the LDAP protocol version 3:

```
com.iona.isp.adapter.LDAP.param.version=3
```

com.iona.isp.adapter.LDAP.params

Obsolete. This property was needed by earlier versions of the Orbix security service, but is now ignored.

com.iona.isp.authz.adapters

Specifies the name of the adapter that is loaded to perform authorization. The adapter name is an arbitrary identifier, *AdapterName*, which is used to construct the names of the properties that configure the adapter—that is, com.iona.isp.authz.adapter.*AdapterName*.class and com.iona.isp.authz.adapter.*AdapterName*.param.filelist. For example:

```
com.iona.isp.authz.adapters=file
com.iona.isp.authz.adapter.file.class=com.iona.security.is2AzAda
   pter.multifile.MultiFileAzAdapter
com.iona.isp.authz.adapter.file.param.filelist=ACLFileListFile;
```

com.iona.isp.authz.adapter.*AdapterName*.class

Selects the authorization adapter class for the *AdapterName* adapter. The following adapter implementations are provided by Orbix:

- `com.iona.security.is2AzAdapter.multifile.MultiFileAzAdapter`—an authorization adapter that enables you to specify multiple ACL files. It is used in conjunction with the `com.iona.isp.authz.adapter.file.param.filelist` property.

For example:

```
com.iona.isp.authz.adapters = file
com.iona.isp.authz.adapter.file.class=com.iona.security.is2AzAda
   pter.multifile.MultiFileAzAdapter
```

com.iona.isp.authz.adapter.*AdapterName*.param.filelist

Specifies the absolute pathname of a file containing a list of ACL files for the *AdapterName* adapter. Each line of the specified file has the following format:

```
[ACLKey=]ACLFileName
```

A file name can optionally be preceded by an ACL key and an equals sign, *ACLKey*=, if you want to select the file by ACL key. The ACL file, *ACLFileName*, is specified using an absolute pathname in the local file format.

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

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

is2.current.server.id

The server ID is an alphanumeric string (excluding spaces) that specifies the current Orbix security service's ID. The server ID is needed for clustering. When a secure application obtains a single sign-on (SSO) token from this Orbix security service, the server ID is embedded into the SSO token. Subsequently, if the SSO token is passed to a *second* Orbix security service instance, the second Orbix security service recognizes that the SSO token originates from the first Orbix security service and delegates security operations to the first Orbix security service.

The server ID is also used to identify replicas in the `cluster.properties` file.

For example, to assign a server ID of `1` to the current Orbix security service:

```
is2.current.server.id=1
```

is2.cluster.properties.filename

Specifies the file that stores the configuration properties for clustering. For example:

```
is2.cluster.properties.filename=C:/is2_config/cluster.properties
```

is2.replication.required

Enables the replication feature of the Orbix security service, which can be used in the context of security service clustering. The possible values are `true` (enabled) and `false` (disabled). When replication is enabled, the security service pushes its cache of SSO data to other servers in the cluster at regular intervals.

Default is `false`.

For example:

```
is2.replication.required=true
```

is2.replication.interval

Specifies the time interval between replication updates to other servers in the security service cluster. The value is specified in units of a second.

Default is 30 seconds.

For example:

```
is2.replication.interval=10
```

is2.replica.selector.classname

If replication is enabled (see `is2.replication.required`), you must set this variable equal to `com.iona.security.replicate.StaticReplicaSelector`.

For example:

```
is2.replica.selector.classname=com.iona.security.replicate.Stati
   cReplicaSelector
```

is2.sso.cache.size

Specifies the maximum cache size (number of user sessions) associated with single sign-on (SSO) feature. The SSO caches user information, including the user's group and role information. If the maximum cache size is reached, the oldest sessions are deleted from the session cache.

No default.

For example:

```
is2.sso.cache.size=1000
```

is2.sso.enabled

Enables the single sign-on (SSO) feature of the Orbix security service. The possible values are `yes` (enabled) and `no` (disabled).

Default is `yes`.

For example:

```
is2.sso.enabled=yes
```

is2.sso.remote.token.cached

In a federated scenario, this variable enables caching of token data for tokens that originate from another security service in the federated cluster. When this variable is set to `true`, a security service need contact another

security service in the cluster, only when the remote token is authenticated for the first time. For subsequent token authentications, the token data for the remote token can be retrieved from the local cache.

Default is `false`.

is2.sso.session.idle.timeout

Sets the session idle time-out in units of seconds for the single sign-on (SSO) feature of the Orbix security service. A zero value implies no time-out.

If a user logs on to the Orbix Security Framework (supplying username and password) with SSO enabled, the Orbix security service returns an SSO token for the user. The next time the user needs to access a resource, there is no need to log on again because the SSO token can be used instead. However, if no secure operations are performed using the SSO token for the length of time specified in the idle time-out, the SSO token expires and the user must log on again.

Default is `0` (no time-out).

For example:

```
is2.sso.session.idle.timeout=0
```

is2.sso.session.timeout

Sets the absolute session time-out in units of seconds for the single sign-on (SSO) feature of the Orbix security service. A zero value implies no time-out.

This is the maximum length of time since the time of the original user login for which an SSO token remains valid. After this time interval elapses, the session expires irrespective of whether the session has been active or idle. The user must then login again.

Default is `0` (no time-out).

For example:

```
is2.sso.session.timeout=0
```

log4j.configuration

Specifies the log4j configuration filename. You can use the properties in this file to customize the level of debugging output from the Orbix security service. See also "log4j Properties File" on page 626.

For example:

```
log4j.configuration=d:/temp/myconfig.txt
```

Cluster Properties File

Overview

The cluster properties file is used to store properties common to a group of Orbix security service instances that operate as a cluster or federation. This section provides descriptions of all the properties that can be specified in a cluster file.

File location

The location of the cluster properties file is specified by the `is2.cluster.properties.filename` property in the iSF properties file. All of the Orbix security service instances in a cluster or federation must share the same cluster properties file.

List of properties

The following properties can be specified in the cluster properties file:

com.iona.security.common.securityInstanceURL.*<server_ID>*

Specifies the server URL for the *<server_ID>* Orbix security service instance.

When single sign-on (SSO) is enabled together with clustering or federation, the Orbix security service instances use the specified instance URLs to communicate with each other. Because the Orbix security service instances share the same cluster file, they can read each other's URLs and open connections to each other.

The connections between Orbix security service instances are made using the IIOP protocol combined with SSL/TLS. The detailed configuration of the IIOP/TLS endpoint is specified in the Orbix configuration file for each security service in the cluster. Hence, you can discover the host and port used by a particular security service by inspecting the values of the `plugins:security:iiop_tls:host` and `plugins:security:iiop_tls:port` variables from its Orbix configuration. You can use the host and port values to construct the value of the security instance URL.

For example, consider a cluster of three security services, where the first security service (ID=1) is configured as follows:

```
# Orbix Configuration File for service with ID=1
plugins:security_cluster:iiop_tls:addr_list =
   ["+security01:5001", "+security02:5002", "+security03:5003"];
plugins:security:iiop_tls:host = "5001";
plugins:security:iiop_tls:port = "security01";
```

The `plugins:security:iiop_tls:host` and `plugins:security:iiop_tls:port` variables give the host and port of the first service, `server01:5001`. Assuming the host and port for the second and third services are `server02:5002` and `server03:5003` respectively, you would configure the security instance URLs as follows:

```
# Advertise the locations of the security services in the cluster.
com.iona.security.common.securityInstanceURL.1=corbaloc:it_iiops:1.2@security01:5001/IT_Security
   Service
com.iona.security.common.securityInstanceURL.2=corbaloc:it_iiops:1.2@security02:5002/IT_Security
   Service
com.iona.security.common.securityInstanceURL.3=corbaloc:it_iiops:1.2@security03:5003/IT_Security
   Service
```

com.iona.security.common.replicaURL.*<server_ID>*

A comma-separated list of URLs for the other security services to which this service replicates its SSO token data. In Orbix, the URLs for the other security services are normally specified in a `corbaloc` format.

For example, to configure the first service in a cluster (ID=1) to replicate its SSO token data to the second service (with address, `server02:5002`) and the third service (with address, `server02:5002`) in the cluster, you would add the following line to the `cluster.properties` file:

```
# Configure replication between security services.
com.iona.security.common.replicaURL.1=corbaloc:it_iiops:1.2@security02:5002/IT_SecurityService,c
   orbaloc:it_iiops:1.2@security03:5003/IT_SecurityService
```

log4j Properties File

Overview

The log4j properties file configures log4j logging for your Orbix security service. This section describes a minimal set of log4j properties that can be used to configure basic logging.

log4j documentation

For complete log4j documentation, see the following Web page:

http://jakarta.apache.org/log4j/docs/documentation.html

File location

The location of the log4j properties file is specified by the `log4j.configuration` property in the iSF properties file. For ease of administration, different Orbix security service instances can optionally share a common log4j properties file.

List of properties

To give you some idea of the capabilities of log4j, the following is an incomplete list of properties that can be specified in a log4j properties file:

log4j.appender.*<AppenderHandle>*

This property specifies a log4j appender class that directs *<AppenderHandle>* logging messages to a particular destination. For example, one of the following standard log4j appender classes could be specified:

- `org.apache.log4j.ConsoleAppender`
- `org.apache.log4j.FileAppender`
- `org.apache.log4j.RollingFileAppender`
- `org.apache.log4j.DailyRollingFileAppender`
- `org.apache.log4j.AsynchAppender`
- `org.apache.log4j.WriterAppender`

For example, to log messages to the console screen for the `A1` appender handle:

```
log4j.appender.A1=org.apache.log4j.ConsoleAppender
```

log4j.appender.*<AppenderHandle>*.layout

This property specifies a log4j layout class that is used to format *<AppenderHandle>* logging messages. One of the following standard log4j layout classes could be specified:

- `org.apache.log4j.PatternLayout`
- `org.apache.log4j.HTMLLayout`
- `org.apache.log4j.SimpleLayout`
- `org.apache.log4j.TTCCLayout`

For example, to use the pattern layout class for log messages processed by the `A1` appender:

```
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
```

log4j.appender.*<AppenderHandle>*.layout.ConversionPattern

This property is used only in conjunction with the `org.apache.log4j.PatternLayout` class (when specified by the `log4j.appender.`*`<AppenderHandle>`*`.layout` property) to define the format of a log message.

For example, you can specify a basic conversion pattern for the `A1` appender as follows:

```
log4j.appender.A1.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n
```

log4j.rootCategory

This property is used to specify the logging level of the root logger and to associate the root logger with one or more appenders. The value of this property is specified as a comma separated list as follows:

```
<LogLevel>, <AppenderHandle01>, <AppenderHandle02>, ...
```

The logging level, *<LogLevel>*, can have one of the following values:

- `DEBUG`
- `INFO`
- `WARN`
- `ERORR`

- FATAL

An appender handle is an arbitrary identifier that associates a logger with a particular logging destination.

For example, to select all messages at the `DEBUG` level and direct them to the `A1` appender, you can set the property as follows:

```
log4j.rootCategory=DEBUG, A1
```

APPENDIX C

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 29 shows a selection of the attribute types that you are most likely to encounter:

Table 29: *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 29). 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
```

APPENDIX D

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 30 shows how the options affect client and target policies:

Table 30: *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 30: *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.

Action-Role Mapping DTD

This appendix presents the document type definition (DTD) for the action-role mapping XML file.

DTD file

The action-role mapping DTD is shown in Example 58.

Example 58:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT action-name (#PCDATA)>
<!ELEMENT role-name (#PCDATA)>
<!ELEMENT server-name (#PCDATA)>
<!ELEMENT action-role-mapping (server-name, interface+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT interface (name, action-role+)>
<!ELEMENT parameter EMPTY>
<!ATTLIST parameter
   name  CDATA  #REQUIRED
   value CDATA  #REQUIRED
>
<!ELEMENT parameter-control (parameter+, role-name+)>
<!ELEMENT action-role (action-name, parameter-control*,
   role-name+)>
<!ELEMENT allow-unlisted-interfaces (#PCDATA)>
<!ELEMENT secure-system (allow-unlisted-interfaces*,
   action-role-mapping+)>
```

Action-role mapping elements

The elements of the action-role mapping DTD can be described as follows:

`<!ELEMENT action-name (#PCDATA)>`

Specifies the action name to which permissions are assigned. The interpretation of the action name depends on the type of application:

- CORBA server—for IDL operations, the action name corresponds to the GIOP on-the-wire format of the operation name (usually the same as it appears in IDL).

  For IDL attributes, the accessor or modifier action name corresponds to the GIOP on-the-wire format of the attribute accessor or modifier. For example, an IDL attribute, `foo`, would have an accessor, `_get_foo`, and a modifier, `_set_foo`.

- Artix server—for WSDL operations, the action name is equivalent to a WSDL operation name; that is, the *`OperationName`* from a tag, `<operation name="`*`OperationName`*`">`.

`<!ELEMENT action-role (action-name, parameter-control*, role-name+)>`

Groups together a particular action and all of the roles permitted to perform that action.

`<!ELEMENT action-role-mapping (server-name, interface+)>`

Contains all of the permissions that apply to a particular server application.

`<!ELEMENT allow-unlisted-interfaces (#PCDATA)>`

Specifies the default access permissions that apply to interfaces not explicitly listed in the action-role mapping file. The element contents can have the following values:

- `true`—for any interfaces not listed, access to all of the interfaces' actions is allowed for all roles. If the remote user is unauthenticated (in the sense that no credentials are sent by the client), access is also allowed.

Note: However, if `<allow-unlisted-interfaces>` is `true` and a particular interface is listed, then only the actions explicitly listed within that interface's `interface` element are accessible. Unlisted actions from the listed interface are not accessible.

- `false`—for any interfaces not listed, access to all of the interfaces' actions is denied for all roles. Unauthenticated users are also denied access.

Default is `false`.

`<!ELEMENT interface (name, action-role+)>`

In the case of a CORBA server, the `interface` element contains all of the access permissions for one particular IDL interface.

In the case of an Artix server, the `interface` element contains all of the access permissions for one particular WSDL port type.

You can also use the wildcard, *, to match any number of contiguous characters in an interface name.

`<!ELEMENT name (#PCDATA)>`

Within the scope of an `interface` element, identifies the interface (IDL interface or WSDL port type) with which permissions are being associated. The format of the interface name depends on the type of application, as follows:

- CORBA server—the `name` element identifies the IDL interface using the interface's OMG repository ID. The repository ID normally consists of the characters `IDL:` followed by the fully scoped name of the interface (using `/` instead of `::` as the scoping character), followed by the characters `:1.0`. Hence, the `Simple::SimpleObject` IDL interface is identified by the `IDL:Simple/SimpleObject:1.0` repository ID.

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

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

- Artix server—the `name` element contains a WSDL port type name, specified in the following format:

  `NamespaceURI``:`*`PortTypeName`*

  The *`PortTypeName`* comes from a tag, `<portType name="`*`PortTypeName`*`">`, defined in the *`NamespaceURI`* namespace.

The *NamespaceURI* is usually defined in the <definitions targetNamespace="*NamespaceURI*" ...> tag of the WSDL contract.

```
<!ELEMENT parameter EMPTY>
<!ATTLIST parameter
   name  CDATA  #REQUIRED
   value CDATA  #REQUIRED
>
```

The <parameter> element is used in conjunction with the action-role mapping feature to restrict user access to an action. A user role is allowed to access an action only if the parameter specified by the name attribute has the value specified by the value attribute.

Note: By default, the <parameter> and <parameter-control> tags only have an effect for the CFR service. Extending this feature to work with other services requires the IONA ART plug-in development kit.

```
<!ELEMENT parameter-control (parameter+, role-name+)>
```

Specifies access control based on the values of certain parameters of the associated action. The role names listed within the <parameter-control> element are granted access to the enclosing action *only* if the parameters take the values specified by the <parameter> tags.

```
<!ELEMENT role-name (#PCDATA)>
```

Specifies a role to which permission is granted. The role name can be any role that belongs to the server's Artix authorization realm (for CORBA bindings, the realm name is specified by the plugins:gsp:authorization_realm configuration variable; for SOAP bindings, the realm name is specified by the plugins:asp:authorization_realm configuration variable) or to the IONAGlobalRealm realm. The roles themselves are defined in the security server backend; for example, in a file adapter file or in an LDAP backend.

```
<!ELEMENT secure-system (allow-unlisted-interfaces*,
   action-role-mapping+)>
```

The outermost scope of an action-role mapping file groups together a collection of action-role-mapping elements.

```
<!ELEMENT server-name (#PCDATA)>
```

The `server-name` element specifies the configuration scope (that is, the ORB name) used by the server in question. This is normally the value of the `-ORBname` parameter passed to the server executable on the command line.

You can also use the wildcard, *, to match any number of contiguous characters in a configuration scope name.

APPENDIX F

OpenSSL Utilities

The `openssl` *program consists of a large number of utilities that have been combined into one program. This appendix describes how you use the* `openssl` *program with Orbix when managing X.509 certificates and private keys.*

In this appendix

This appendix contains the following sections:

Using OpenSSL Utilities

The OpenSSL package

Orbix ships a version of the OpenSSL program that is available with Eric Young's openssl package. OpenSSL is a publicly available implementation of the SSL protocol. Consult "License Issues" on page 665 for information about the copyright terms of OpenSSL.

Note: For complete documentation of the OpenSSL utilities, consult the documentation at the OpenSSL web site `http://www.openssl.org/docs`.

Command syntax

An `openssl` command line takes the following form:

`openssl` *utility arguments*

For example:

```
openssl x509 -in OrbixCA -text
```

The `openssl` utilities

This appendix describes four `openssl` utilities:

`x509`	Manipulates X.509 certificates.
`req`	Creates and manipulates certificate signing requests, and self-signed certificates.
`rsa`	Manipulates RSA private keys.
`ca`	Implements a Certification Authority (CA).

The `-help` option

To get a list of the arguments associated with a particular command, use the `-help` option as follows:

`openssl` *utility* `-help`

For example:

```
openssl x509 -help
```

The x509 Utility

Purpose of the `x509` utility

In Orbix the `x509` utility is mainly used for:

- Printing text details of certificates you wish to examine.
- Converting certificates to different formats.

Options

The options supported by the openssl `x509` utility are as follows:

```
-inform arg        - input format - default PEM
                   (one of DER, NET or PEM)
-outform arg       - output format - default PEM
                   (one of DER, NET or PEM
-keyform arg       - private key format - default PEM
-CAform arg        - CA format - default PEM
-CAkeyform arg     - CA key format - default PEM
-in arg            - input file - default stdin
-out arg           - output file - default stdout
-serial            - print serial number value
-hash              - print serial number value
-subject           - print subject DN
-issuer            - print issuer DN
-startdate         - notBefore field
-enddate           - notAfter field
-dates             - both Before and After dates
-modulus           - print the RSA key modulus
-fingerprint       - print the certificate fingerprint
-noout             - no certificate output
-days arg          - How long till expiry of a signed certificate
                   - def 30 days
-signkey arg       - self sign cert with arg
-x509toreq         - output a certification request object
-req               - input is a certificate request, sign and
                   output
-CA arg            - set the CA certificate, must be PEM format
```

```
-CAkey arg          - set the CA key, must be PEM format. If missing
                    it is assumed to be in the CA file
-CAcreateserial     - create serial number file if it does not exist
-CAserial           - serial file
-text               - print the certificate in text form
-C                  - print out C code forms
-md2/-md5/-sha1/    - digest to do an RSA sign with
-mdc2
```

Using the x509 utility

To print the text details of an existing PEM-format X.509 certificate, use the `x509` utility as follows:

```
openssl x509 -in MyCert.pem -inform PEM -text
```

To print the text details of an existing DER-format X.509 certificate, use the `x509` utility as follows:

```
openssl x509 -in MyCert.der -inform DER -text
```

To change a certificate from PEM format to DER format, use the `x509` utility as follows:

```
openssl x509 -in MyCert.pem -inform PEM -outform DER -out
   MyCert.der
```

The req Utility

Purpose of the x509 utility

The req utility is used to generate a self-signed certificate or a certificate signing request (CSR). A CSR contains details of a certificate to be issued by a CA. When creating a CSR, the req command prompts you for the necessary information from which a certificate request file and an encrypted private key file are produced. The certificate request is then submitted to a CA for signing.

If the -nodes (no DES) parameter is not supplied to req, you are prompted for a pass phrase which will be used to protect the private key.

Note: It is important to specify a validity period (using the -days parameter). If the certificate expires, applications that are using that certificate will not be authenticated successfully.

Options

The options supported by the openssl req utility are as follows:

-inform arg	input format - one of DER TXT PEM
-outform	arg output format - one of DER TXT PEM
-in arg	inout file
-out arg	output file
-text	text form of request
-noout	do not output REQ
-verify	verify signature on REQ
-modulus	RSA modulus
-nodes	do not encrypt the output key
-key file	use the private key contained in file
-keyform arg	key file format
-keyout arg	file to send the key to
-newkey rsa:bits	generate a new RSA key of ‘bits’ in size
-newkey dsa:file	generate a new DSA key, parameters taken from CA in ‘file’
-[digest]	Digest to sign with (md5, sha1, md2, mdc2)
-config file	request template file

-new	new request
-x509	output an x509 structure instead of a certificate req. (Used for creating self signed certificates)
-days	number of days an x509 generated by -x509 is valid for
-asn1-kludge	Output the 'request' in a format that is wrong but some CA's have been reported as requiring [It is now always turned on but can be turned off with -no-asn1-kludge]

Using the req Utility

To create a self-signed certificate with an expiry date a year from now, the req utility can be used as follows to create the certificate CA_cert.pem and the corresponding encrypted private key file CA_pk.pem:

```
openssl req -config ssl_conf_path_name -days 365
-out CA_cert.pem -new -x509 -keyout CA_pk.pem
```

This following command creates the certificate request MyReq.pem and the corresponding encrypted private key file MyEncryptedKey.pem:

```
openssl req -config ssl_conf_path_name -days 365
-out MyReq.pem -new -keyout MyEncryptedKey.pem
```

The rsa Utility

Purpose of the `rsa` utility

The `rsa` command is a useful utility for examining and modifying RSA private key files. Generally RSA keys are stored encrypted with a symmetric algorithm using a user-supplied pass phrase. The OpenSSL `req` command prompts the user for a pass phrase in order to encrypt the private key. By default, `req` uses the triple DES algorithm. The `rsa` command can be used to change the password that protects the private key and to convert the format of the private key. Any `rsa` command that involves reading an encrypted `rsa` private key will prompt for the PEM pass phrase used to encrypt it.

Options

The options supported by the openssl `rsa` utility are as follows:

`-inform arg`	`input format - one of DER NET PEM`
`-outform arg`	`output format - one of DER NET PEM`
`-in arg`	`inout file`
`-out arg`	`output file`
`-des`	`encrypt PEM output with cbc des`
`-des3`	`encrypt PEM output with ede cbc des using 168 bit key`
`-text`	`print the key in text`
`-noout`	`do not print key out`
`-modulus`	`print the RSA key modulus`

Using the rsa Utility

Converting a private key to PEM format from DER format involves using the `rsa` utility as follows:

```
openssl rsa -inform DER -in MyKey.der -outform PEM -out MyKey.pem
```

Changing the pass phrase which is used to encrypt the private key involves using the `rsa` utility as follows:

```
openssl rsa -inform PEM -in MyKey.pem -outform PEM -out MyKey.pem
   -des3
```

Removing encryption from the private key (which is not recommended) involves using the `rsa` command utility as follows:

```
openssl rsa -inform PEM -in MyKey.pem -outform PEM -out MyKey2.pem
```

Note: Do not specify the same file for the `-in` and `-out` parameters, because this can corrupt the file.

The ca Utility

Purpose of the ca utility

You can use the ca utility create X.509 certificates by signing existing signing requests. It is imperative that you check the details of a certificate request before signing. Your organization should have a policy with respect to the issuing of certificates.

The ca utility is used to sign certificate requests thereby creating a valid X.509 certificate which can be returned to the request submitter. It can also be used to generate Certificate Revocation Lists (CRLS). For information on the ca -policy and -name options, refer to "The OpenSSL Configuration File" on page 655.

Creating a new CA

To create a new CA using the openssl ca utility, two files (serial and index.txt) need to be created in the location specified by the openssl configuration file that you are using.

Options

The options supported by the openssl ca utility are as follows:

```
-verbose          - Talk alot while doing things
-config file      - A config file
-name arg         - The particular CA definition to use
-gencrl           - Generate a new CRL
-crldays days     - Days is when the next CRL is due
-crlhours hours   - Hours is when the next CRL is due
-days arg         - number of days to certify the certificate for
-md arg           - md to use, one of md2, md5, sha or sha1
-policy arg       - The CA 'policy' to support
-keyfile arg      - PEM private key file
-key arg          - key to decode the private key if it is
                    encrypted
-cert             - The CA certificate
-in file          - The input PEM encoded certificate request(s)
-out file         - Where to put the output file(s)
-outdir dir       - Where to put output certificates
```

```
-infiles....      - The last argument, requests to process
-spkac file       - File contains DN and signed public key and
                  challenge
-preserveDN       - Do not re-order the DN
-batch            - Do not ask questions
-msie_hack        - msie modifications to handle all thos
                     universal strings
```

Note: Most of the above parameters have default values as defined in `openssl.cnf`.

Using the ca Utility

Converting a private key to PEM format from DER format involves using the ca utility as shown in the following example. To sign the supplied CSR `MyReq.pem` to be valid for 365 days and create a new X.509 certificate in PEM format, use the ca utility as follows:

```
openssl ca -config ssl_conf_path_name -days 365
-in MyReq.pem -out MyNewCert.pem
```

The OpenSSL Configuration File

Overview

A number of OpenSSL commands (for example, `req` and `ca`) take a `-config` parameter that specifies the location of the openssl configuration file. This section provides a brief description of the format of the configuration file and how it applies to the `req` and `ca` commands. An example configuration file is listed at the end of this section.

Structure of `openssl.cnf`

The `openssl.cnf` configuration file consists of a number of sections that specify a series of default values that are used by the openssl commands.

In this section

This section contains the following subsections:

[req] Variables

Overview of the variables

The `req` section contains the following variables:

```
default_bits = 1024
default_keyfile = privkey.pem
distinguished_name = req_distinguished_name
attributes = req_attributes
```

`default_bits` configuration variable

The `default_bits` variable is the default RSA key size that you wish to use. Other possible values are 512, 2048, and 4096.

`default_keyfile` configuration variable

The `default_keyfile` variable is the default name for the private key file created by `req`.

`distinguished_name` configuration variable

The `distinguished_name` variable specifies the section in the configuration file that defines the default values for components of the distinguished name field. The `req_attributes` variable specifies the section in the configuration file that defines defaults for certificate request attributes.

[ca] Variables

Choosing the CA section

You can configure the file `openssl.cnf` to support a number of CAs that have different policies for signing CSRs. The `-name` parameter to the `ca` command specifies which CA section to use. For example:

```
openssl ca -name MyCa ...
```

This command refers to the CA section `[MyCa]`. If `-name` is not supplied to the `ca` command, the CA section used is the one indicated by the `default_ca` variable. In the "Example openssl.cnf File" on page 659, this is set to `CA_default` (which is the name of another section listing the defaults for a number of settings associated with the `ca` command). Multiple different CAs can be supported in the configuration file, but there can be only one default CA.

Overview of the variables

Possible `[ca]` variables include the following

```
dir: The location for the CA database
     The database is a simple text database containing the
        following tab separated fields:

status:    A value of 'R' - revoked, 'E' -expired or 'V' valid
issued date: When the certificate was certified
revoked date: When it was revoked, blank if not revoked
serial number: The certificate serial number
certificate: Where the certificate is located
CN: The name of the certificate
```

The `serial number` field should be unique, as should the `CN`/`status` combination. The `ca` utility checks these at startup.

```
certs: This is where all the previously issued certificates are
   kept
```

[policy] Variables

Choosing the policy section

The policy variable specifies the default policy section to be used if the `-policy` argument is not supplied to the `ca` command. The CA policy section of a configuration file identifies the requirements for the contents of a certificate request which must be met before it is signed by the CA.

There are two policy sections defined in the "Example openssl.cnf File" on page 659: `policy_match` and `policy_anything`.

Example policy section

The `policy_match` section of the example `openssl.cnf` file specifies the order of the attributes in the generated certificate as follows:

```
countryName
stateOrProvinceName
organizationName
organizationalUnitName
commonName
emailAddress
```

The `match` policy value

Consider the following value:

```
countryName = match
```

This means that the country name must match the CA certificate.

The `optional` policy value

Consider the following value:

```
organisationalUnitName  = optional
```

This means that the `organisationalUnitName` does not have to be present.

The `supplied` policy value

Consider the following value:

```
commonName = supplied
```

This means that the `commonName` must be supplied in the certificate request.

Example openssl.cnf File

Listing

The following listing shows the contents of an example `openssl.cnf` configuration file:

```
##################################################################
# openssl example configuration file.
# This is mostly used for generation of certificate requests.
##################################################################
[ ca ]
default_ca= CA_default          # The default ca section
##################################################################

[ CA_default ]

dir=/opt/iona/OrbixSSL1.0c/certs # Where everything is kept

certs=$dir # Where the issued certs are kept
crl_dir= $dir/crl # Where the issued crl are kept
database= $dir/index.txt # database index file
new_certs_dir= $dir/new_certs # default place for new certs
certificate=$dir/CA/OrbixCA # The CA certificate
serial= $dir/serial # The current serial number
crl= $dir/crl.pem # The current CRL
private_key= $dir/CA/OrbixCA.pk # The private key
RANDFILE= $dir/.rand # private random number file
default_days= 365 # how long to certify for
default_crl_days= 30 # how long before next CRL
default_md= md5 # which message digest to use
preserve= no # keep passed DN ordering

# A few different ways of specifying how closely the request
   should
# conform to the details of the CA

policy= policy_match

# For the CA policy

[policy_match]
countryName= match
stateOrProvinceName= match
organizationName= match
organizationalUnitName= optional
commonName= supplied
```

```
emailAddress= optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types

[ policy_anything ]
countryName = optional
stateOrProvinceName= optional
localityName= optional
organizationName = optional
organizationalUnitName = optional
commonName= supplied
emailAddress= optional

[ req ]
default_bits = 1024
default_keyfile= privkey.pem
distinguished_name = req_distinguished_name
attributes = req_attributes

[ req_distinguished_name ]
countryName= Country Name (2 letter code)
countryName_min= 2
countryName_max = 2
stateOrProvinceName= State or Province Name (full name)
localityName = Locality Name (eg, city)
organizationName = Organization Name (eg, company)
organizationalUnitName  = Organizational Unit Name (eg, section)
commonName = Common Name (eg. YOUR name)
commonName_max = 64
emailAddress = Email Address
emailAddress_max = 40

[ req_attributes ]
challengePassword = A challenge password
challengePassword_min = 4
challengePassword_max = 20
unstructuredName= An optional company name
```

APPENDIX G

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, the configuration repository (CFR), 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).

WARNING: Do *not* use the CFR for the configuration of security information in this release. The CFR could be modified by unauthorized clients which would compromise secure application configuration.

File based configuration must be used for secure applications.

APPENDIX H

License Issues

This appendix contains the text of licenses relevant to Orbix.

In this appendix

This appendix contains the following section:

OpenSSL License

Overview

The licence agreement for the usage of the OpenSSL command line utility shipped with Orbix SSL/TLS is as follows:

```
LICENSE ISSUES
==============
  The OpenSSL toolkit stays under a dual license, i.e. both the conditions of
  the OpenSSL License and the original SSLeay license apply to the toolkit.
  See below for the actual license texts. Actually both licenses are BSD-style
  Open Source licenses. In case of any license issues related to OpenSSL
  please contact openssl-core@openssl.org.

  OpenSSL License
  ---------------

/* ====================================================================
* Copyright (c) 1998-1999 The OpenSSL Project.  All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
* 1. Redistributions of source code must retain the above copyright
*    notice, this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in
*    the documentation and/or other materials provided with the
*    distribution.
*
* 3. All advertising materials mentioning features or use of this
*    software must display the following acknowledgment:
*    "This product includes software developed by the OpenSSL Project
*    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
*
* 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
*    endorse or promote products derived from this software without
*    prior written permission. For written permission, please contact
*    openssl-core@openssl.org.
*
* 5. Products derived from this software may not be called "OpenSSL"
*    nor may "OpenSSL" appear in their names without prior written
*    permission of the OpenSSL Project.
```

```
*
* 6. Redistributions of any form whatsoever must retain the following
*    acknowledgment:
*    "This product includes software developed by the OpenSSL Project
*    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
*
* THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
* PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
* OF THE POSSIBILITY OF SUCH DAMAGE.
* ====================================================================
*
* This product includes cryptographic software written by Eric Young
* (eay@cryptsoft.com).  This product includes software written by Tim
* Hudson (tjh@cryptsoft.com).
*
*/

Original SSLeay License
-----------------------

/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
* All rights reserved.
*
* This package is an SSL implementation written
* by Eric Young (eay@cryptsoft.com).
* The implementation was written so as to conform with Netscapes SSL.
*
* This library is free for commercial and non-commercial use as long as
* the following conditions are aheared to.  The following conditions
* apply to all code found in this distribution, be it the RC4, RSA,
* lhash, DES, etc., code; not just the SSL code.  The SSL documentation
* included with this distribution is covered by the same copyright terms
* except that the holder is Tim Hudson (tjh@cryptsoft.com).
*
* Copyright remains Eric Young's, and as such any Copyright notices in
* the code are not to be removed.
* If this package is used in a product, Eric Young should be given attribution
* as the author of the parts of the library used.
```

```
* This can be in the form of a textual message at program startup or
* in documentation (online or textual) provided with the package.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the copyright
*    notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in the
*    documentation and/or other materials provided with the distribution.
* 3. All advertising materials mentioning features or use of this software
*    must display the following acknowledgement:
*    "This product includes cryptographic software written by
*     Eric Young (eay@cryptsoft.com)"
*    The word 'cryptographic' can be left out if the rouines from the library
*    being used are not cryptographic related :-).
* 4. If you include any Windows specific code (or a derivative thereof) from
*    the apps directory (application code) you must include an acknowledgement:
*    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
*
* THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND
* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*
* The licence and distribution terms for any publically available version or
* derivative of this code cannot be changed.  i.e. this code cannot simply be
* copied and put under another distribution licence
* [including the GNU Public Licence.]
*/
```

Index

D

E

F

G

H

I

J

K

N

O

P

Q

R

S

V

W

X