BEA Tuxedo®

BEA Tuxedo CORBA アプリケーションのセキュリティ機能

BEA Tuxedo 8.1
2003 年 1 月

Copyright

Restricted Rights Legend

Trademarks or Service Marks

目次

5. リンク・レベルの暗号化のコンフィギュレーション

6. SSL プロトコルのコンフィギュレーション

7. 認証のコンフィギュレーション

8. シングル・サイン・オンのコンフィギュレーション

このマニュアルについて

このマニュアルでは、BEA Tuxedo セキュリティ機能に関連する概念、セキュリティ機能を使用して CORBA アプリケーションを保護する方法、および CORBA セキュリティ・サービスのアプリケーション・プログラミング・インターフェイス (API) を使用する方法について説明します。

注記　BEA Tuxedo のリリース 8.0 には、アプリケーション・トランザクション・モニタ・インターフェイス (ATMI) アプリケーションおよび CORBA アプリケーションをビルドできる環境が用意されています。このマニュアルでは、CORBA アプリケーションでセキュリティをインプリメントする方法について説明します。ATMI アプリケーションでセキュリティをインプリメントする方法については、『BEA Tuxedo のセキュリティ機能』を参照してください。

このマニュアルでは、以下の内容について説明します。

- 第 1 章 「CORBA セキュリティ機能の概要」では、BEA Tuxedo 製品の CORBA セキュリティ機能について概説します。
- 第 2 章 「SSL 技術の概要」では、PKI (Public Key Infrastructure) に関連する概念について説明します。
- 第 3 章 「CORBA のセキュリティの基本概念」では、CORBA セキュリティ・サービスの機能、およびこの機能をインプリメントするための開発および管理プロセスについて説明します。
- 第 4 章 「公開鍵によるセキュリティ機能の管理」では、PKI を設定して、セキュア・ソケット・レイヤ (SSL) プロトコルおよび証明書による認証を使用する CORBA アプリケーションとやり取りする方法について説明します。

- 第 5 章 「リンク・レベルの暗号化のコンフィギュレーション」では、リンク・レベルの暗号化 (LLE) 用に UBBCONFIG ファイルのパラメータを設定する方法について説明します。
- 第 6 章 「SSL プロトコルのコンフィギュレーション」では、セキュア・ソケット・レイヤ (SSL) プロコトルおよび証明書による認証で使用できるように、IIOP リスナ / ハンドラまたは CORBA C++ ORB をコンフィギュレーションする方法について説明します。
- 第 7 章 「認証のコンフィギュレーション」では、CORBA アプリケーションで認証を使用するために必要なコンフィギュレーション・タスクについて説明します。
- 第 8 章 「シングル・サイン・オンのコンフィギュレーション」では、CORBA アプリケーションで信頼性のある接続プールを使用するために必要なコンフィギュレーション・タスクについて説明します。
- 第 9 章 「セキュリティ・プラグインのコンフィギュレーション」では、CORBA 環境でセキュリティ・プラグインを登録する方法について説明します。
- 第 10 章 「セキュリティをインプリメントする CORBA アプリケーション」では、ブートストラップ処理オプションのしくみと、CORBA アプリケーションにパスワードおよび証明書による認証をインプリメントする方法について説明します。
- 第 11 章 「CORBA サンプル・アプリケーションのビルドと実行」では、Security および Secure Simpapp サンプル・アプリケーションをビルドおよび実行する方法について説明します。
- 第 12 章 「トラブルシューティング」では、CORBA アプリケーションのセキュリティに関する問題を解決するための、トラブルシューティングのヒントを紹介します。
- 第 13 章 「CORBA セキュリティ API」では、CORBA アプリケーションのセキュリティ・モデル、およびそのセキュリティ・モデルで機能するコンポーネントについて説明します。
- 第 14 章 「セキュリティ・モジュール」では、CORBA セキュリティ・サービスで使用するモジュール用の Object Management Group (OMG) インターフェイス定義言語 (IDL) について説明します。

- 第 15 章「C++ セキュリティ・リファレンス」では、C++ メソッドについて説明します。
- 第 16 章「Java セキュリティ・リファレンス」では、Java メソッドについて説明します。
- 第 17 章「オートメーション・セキュリティ・リファレンス」では、オートメーション・メソッドについて説明します。

対象読者

このマニュアルは、セキュリティを CORBA アプリケーションに統合するプログラマ、および企業内のセキュリティ・インフラストラクチャを設定および管理するシステム管理者を対象としています。

e-docs Web サイト

BEA Tuxedo 製品のマニュアルは BEA 社の Web サイトで参照することができます。BEA ホーム・ページの [製品のドキュメント] をクリックするか、または http://edocs.beasys.co.jp/e-docs/index.html に直接アクセスしてください。

マニュアルの印刷方法

このマニュアルは、ご使用の Web ブラウザで一度に 1 ファイルずつ印刷できます。Web ブラウザの [ファイル] メニューにある [印刷] オプションを使用してください。

このマニュアルの PDF 版は、Web サイト上にあります。また、マニュアルの CD-ROM にも収められています。この PDF を Adobe Acrobat Reader で開くと、マニュアル全体または一部をブック形式で印刷できます。PDF 形式を利用するには、BEA Tuxedo マニュアルのホーム・ページにある [PDF 版] ボタンをクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader をお持ちでない場合は、Adobe Web サイト (http://www.adobe.co.jp/) から無償でダウンロードできます。

関連情報

CORBA、BEA Tuxedo、分散オブジェクト・コンピューティング、トランザクション処理、C++、および Java プログラミングの詳細については、BEA Tuxedo オンライン・マニュアルの「Bibliography」を参照してください。

サポート情報

皆様の BEA Tuxedo マニュアルに対するフィードバックをお待ちしています。ご意見やご質問がありましたら、電子メールで docsupport-jp@beasys.com までお送りください。お寄せいただきましたご意見は、BEA Tuxedo マニュアルの作成および改訂を担当する BEA 社のスタッフが直接検討いたします。

電子メール メッセージには、BEA Tuxedo 8.0 リリースのマニュアルを使用していることを明記してください。

BEA Tuxedo に関するご質問、または BEA Tuxedo のインストールや使用に際して問題が発生した場合は、www.beasys.com の BEA WebSUPPORT を通して BEA カスタマ・サポートにお問い合わせください。カスタマ・サポートへの問い合わせ方法は、製品パッケージに同梱されているカスタマ・サポート・カードにも記載されています。

カスタマ・サポートへお問い合わせの際には、以下の情報をご用意ください。

- お客様のお名前、電子メール・アドレス、電話番号、Fax 番号
- お客様の会社名と会社の住所
- ご使用のマシンの機種と認証コード
- ご使用の製品名とバージョン
- 問題の説明と関連するエラー・メッセージの内容

表記上の規則

このマニュアルでは、以下の表記規則が使用されています。

規則	項目
太字	用語集に定義されている用語を示します。
Ctrl + Tab	2 つ以上のキーを同時に押す操作を示します。
イタリック体	強調またはマニュアルのタイトルを示します。

規則	項目

等幅テキスト	コード・サンプル、コマンドとオプション、データ構造とメンバ、データ型、ディレクトリ、およびファイル名と拡張子を示します。また、キーボードから入力するテキストも等幅テキストで表示します。
	例:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

太字の等幅テキスト	コード内の重要な語を示します。 例: `void` **`commit`** `( )`
斜体の等幅テキスト	コード内の変数を示します。 例: `String` *`expr`*
大文字のテキスト	デバイス名、環境変数、および論理演算子を示します。 例: LPT1 SIGNON OR
{ }	構文の行で、選択肢の組み合わせを示します。かっこは入力しません。

[ ]	構文の行で、オプション項目を示します。かっこは入力しません。
	例:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

|	構文の行で、相互に排他的な選択肢の区切りとして使います。記号は入力しません。

規則	項目
. . .	コマンド・ラインで、以下のいずれかの場合を示します。 ■ コマンド・ラインで、同じ引数を繰り返し使用できることを示します。 ■ 文で、追加のオプション引数が省略されていることを示します。 ■ 追加のパラメータ、値、またはその他の情報を入力できることを示します。 記号は入力しません。 例: `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
. . .	コード例または構文の行で、項目が省略されていることを示します。記号は入力しません。

第 I 部 セキュリティの概念

1 CORBA セキュリティ機能の概要

ここでは、以下の内容について説明します。

- CORBA セキュリティ機能
- CORBA セキュリティ環境
- CORBA セキュリティ環境のシングル・サイン・オン
- BEA Tuxedo セキュリティ SPI

注記　BEA Tuxedo のリリース 8.0 には、アプリケーション・トランザクション・モニタ・インターフェイス (ATMI) アプリケーションおよび CORBA アプリケーションをビルドできる環境が用意されています。このマニュアルでは、CORBA アプリケーションでセキュリティをインプリメントする方法について説明します。ATMI アプリケーションでセキュリティをインプリメントする方法については、『BEA Tuxedo のセキュリティ機能』を参照してください。

CORBA セキュリティ機能

セキュリティとは、コンピュータ内のデータまたはコンピュータ間で送受信されるデータが損なわれないことを保証する技術のことです。ほとんどのセキュリティ機能では、証明資料およびデータの暗号化を使用してセキュリ

ティを実現します。証明資料とは、秘密の文字列であり、ユーザはこれを入力することにより特定のプログラムやシステムにアクセスできます。データの暗号化とは、解釈不能な形式にデータを変換することです。

電子商取引などで使用される分散アプリケーションには、悪質なユーザがデータを傍受したり、操作を中断したり、不正な情報を入力できるアクセス・ポイントが多数あります。ビジネスがより広い範囲に分散されるほど、こうした悪質なユーザによる攻撃を受けやすくなります。したがって、このようなアプリケーションの基盤となる分散型のコンピューティング・ソフトウェア、つまりミドルウェアは、セキュリティ機能を備えている必要があります。

BEA Tuxedo 製品の CORBA セキュリティ機能を利用すると、クライアント・アプリケーションとサーバ・アプリケーションの間で安全な接続を確立できます。以下の機能があります。

- CORBA C++ および Java クライアント・アプリケーションが BEA Tuxedo ドメインにアクセスする場合の認証。認証には、ユーザ名 / パスワードの組み合わせを使用する標準的な方法と、X.509 デジタル証明書で ID をサーバ・アプリケーションに提示する方法があります。

- リンク・レベルの暗号化 (LLE) またはセキュア・ソケット・レイヤ (SSL) プロトコルによるデータの整合性と機密性。CORBA C++ および Java クライアント・アプリケーションは、BEA Tuxedo ドメインと SSL セッションを確立できます。BEA Tuxedo クライアント・アプリケーションでは、LLE を使用してブリッジとドメインの間のネットワーク・トラフィックを保護します。

- WebLogic Enterprise Connectivity を使用した BEA WebLogic Server™ 環境と CORBA 環境の間のシングル・サイン・オン環境。この機能を使用すると、信頼性のある接続プールを構成するネットワーク接続を介して、要求元の WebLogic Server ユーザに関するセキュリティ情報を BEA Tuxedo ドメインに伝達できます。

- 認証、認可、監査、および公開鍵によるセキュリティ機能を提供するセキュリティ・メカニズムを統合するために使用できるセキュリティ・サービス・プロバイダ・インターフェイス (SPI)。セキュリティ・ベンダは、SPI を使用して、サード・パーティのセキュリティ製品を CORBA 環境に統合できます。

- SSL プロトコルおよび X.509 デジタル証明書を利用して、ネットワーク・リンク経由で送信されるメッセージのデータ機密性を保護する PKI (Public Key Infrastructure)。さらに、PKI SPI のセットも用意されています。

CORBA 環境のすべてのセキュリティ機能を利用するには、SSL プロトコル、LLE、および PKI を有効にするライセンスをインストールする必要があります。セキュリティ機能のライセンスのインストール方法については、『BEA Tuxedo システムのインストール』を参照してください。

注記　『BEA Tuxedo CORBA アプリケーションのセキュリティ機能』では、BEA Tuxedo 製品に用意されている CORBA 環境のセキュリティ機能について説明しています。BEA Tuxedo 製品に用意されている ATMI 環境のセキュリティ機能の使用方法については、『BEA Tuxedo のセキュリティ機能』を参照してください。

表 1-1 は、BEA Tuxedo 製品の CORBA セキュリティ機能の特徴をまとめたものです。

表 3-1 CORBA セキュリティ機能

セキュリティ機能	説明	サービス・プロバイダ・インターフェイス (SPI)	デフォルトのインプリメンテーション
認証	ユーザまたはシステム・プロセスに対して指定されている ID を証明し、ID 情報を安全に記録および転送し、必要に応じて ID 情報を利用可能にします。	1 つのインターフェイスとして実装されます。	認証なし、アプリケーション・パスワードを使用、証明書による認証、という 3 つのセキュリティ・レベルが用意されています。
認可	ID およびその他の情報に基づいて、リソースへのアクセスを制御します。	1 つのインターフェイスとして実装されます。	N/A
監査	操作要求とその結果に関する情報を安全に収集、格納、および配布します。	1 つのインターフェイスとして実装されます。	デフォルトの監査機能は、ユーザ・ログ機能 (ULOG) によって実装されます。

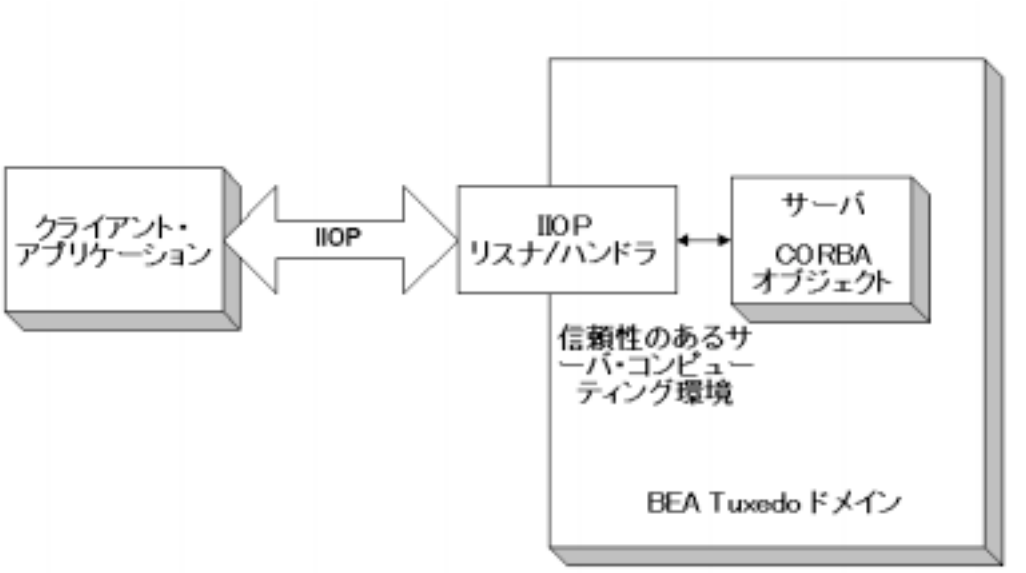
表 3-1 CORBA セキュリティ機能 (続き)

セキュリティ機能	説明	サービス・プロバイダ・インターフェイス (SPI)	デフォルトのインプリメンテーション
リンク・レベルの暗号化	対称鍵による暗号化を使用して、CORBA アプリケーション内のマシン間をつなぐネットワーク・リンク上で送受信されるメッセージのデータ機密性を保護します。	N/A	RC4 形式の対称鍵による暗号化
セキュア・ソケット・レイヤ (SSL) プロトコル	対称鍵による暗号化を使用して、BEA Tuxedo ドメイン間をつなぐネットワーク・リンク上で送受信されるメッセージのデータ機密性を保護します。	N/A	SSL バージョン 3.0 プロトコル
シングル・サイン・オン	WebLogic Server ユーザの ID を BEA Tuxedo ドメインに伝達します。	N/A	N/A
公開鍵によるセキュリティ機能	公開鍵 (または非対称鍵) 暗号化を使用して、リモート・クライアント・アプリケーションと IIOP リスナ / ハンドラ間をつなぐネットワーク・リンク上で送受信されるメッセージのデータ機密性を保護します。X.509 デジタル証明書に基づく相互認証に対応した SSL バージョン 3.0 に準拠しています。	次のインターフェイスとして実装されます。 ■ 公開鍵の初期化 ■ 鍵管理 ■ 証明ルックアップ ■ 証明解析 ■ 証明書の検証 ■ 証明資料のマッピング	デフォルトの公開鍵セキュリティでは、次のアルゴリズムがサポートされます。 ■ 鍵暗号用の RSA ■ バルク暗号化用の DES とそれを改訂した RC2 および RC4 ■ メッセージ・ダイジェスト用の MD5 および SHA

CORBA セキュリティ環境

BEA Tuxedo CORBA 環境などの分散型のエンタープライズ・ミドルウェア環境で、エンド・ツー・エンドの相互認証を直接行う場合、特に、長時間の接続用に最適化されたセキュリティ・メカニズムでは、大幅にコストがかかります。プリンシパルから各サーバ・アプリケーションに対して直接ネットワーク接続を確立したり、サービス要求の処理時に複数の認証メッセージを交換および検証するのは、非効率的です。代わりに、BEA Tuxedo 製品の CORBA アプリケーションは、図 3-1 のような、高信頼性委譲型認証モデルを実装しています。

図 3-1 高信頼性委譲型モデル



高信頼性委譲型モデルでは、プリンシパル (一般にはクライアント・アプリケーションのユーザ) は信頼性のあるシステム・ゲートウェイ・プロセスに対して認証を行います。CORBA アプリケーションの場合、信頼性のあるシ

ステム・ゲートウェイ・プロセスは IIOP リスナ / ハンドラです。認証が成功すると、セキュリティ・トークンが開始元プリンシパルに割り当てられます。セキュリティ・トークンとは、プロセス間の転送に適したオペークなデータ構造体です。

IIOP リスナ / ハンドラは、認証済みのプリンシパルから要求を受信すると、認証および監査用にプリンシパルのセキュリティ・トークンをアタッチして、ターゲット・サーバ・アプリケーションに送ります。

高信頼性委譲型モデルでは、IIOP リスナ / ハンドラは BEA Tuxedo ドメインの認証ソフトウェアがプリンシパルの ID を確認することを前提にして、適切なセキュリティ・トークンを生成します。サーバ・アプリケーションは、IIOP リスナ / ハンドラが正しいセキュリティ・トークンをアタッチすることを前提にしています。また、サーバ・アプリケーションは、プリンシパルの要求に関わるほかのサーバ・アプリケーションがセキュリティ・トークンを安全に転送することを前提にしています。

開始元のクライアント・アプリケーションと IIOP リスナ / ハンドラの間のセッションは、以下のように確立されます。

1. クライアント・アプリケーションが BEA Tuxedo ドメイン内のオブジェクトにアクセスする場合、クライアント・アプリケーションはユーザ名およびパスワードまたは X.509 デジタル証明書を使用して、IIOP リスナ / ハンドラとの接続を認証します。

2. プリンシパルと IIOP リスナ / ハンドラの間で、セキュリティが関連付けられます (セキュリティ・コンテキスト)。このセキュリティ・コンテキストを使用して、BEA Tuxedo ドメイン内のオブジェクトへのアクセスが制御されます。

   IIOP リスナ / ハンドラは、セキュリティ・コンテキストから認証および監査トークンを取り出します。認証および監査トークンは共に、セキュリティ・コンテキストに関連付けられたプリンシパルの ID を表します。

3. 認証が完了すると、プリンシパルは BEA Tuxedo ドメイン内のオブジェクトを呼び出せます。要求は IIOP 要求にパッケージ化され、IIOP リスナ / ハンドラに転送されます。IIOP リスナ / ハンドラは、確立済みのセキュリティ・コンテキストに要求を関連付けます。

4. IIOP リスナ / ハンドラは、開始元プリンシパルから要求を受信します。

クライアント・アプリケーションと IIOP リスナ / ハンドラ間のメッセージが保護されるかどうかは、CORBA アプリケーションで使用されるセキュリティ技術によって決まります。BEA Tuxedo 製品では、認証情報はデフォルトで暗号化されますが、クライアント・アプリケーションと BEA Tuxedo ドメイン間を送信されるメッセージは暗号化されません。メッセージはクリア・テキストで送信されます。SSL プロトコルを使用すると、メッセージを保護できます。SSL プロトコルをコンフィギュレーションしてメッセージの整合性と機密性を保護する場合、要求はデジタル署名を付けて封印 (暗号化) してから、IIOP リスナ / ハンドラに送信されます。

5. IIOP リスナ / ハンドラは、要求に開始元の認証および監査トークンを付けて、適切なサーバ・アプリケーションに転送します。
6. 要求がサーバ・アプリケーションに届くと、BEA Tuxedo システムは転送された要求元プリンシパルのトークンを調べて、要求を処理するのか拒否するのかを決定します。CORBA セキュリティ機能は、認証インプリメンテーションによる決定に基づいて、要求元プリンシパルがアクセス権限を持たないオブジェクトに対する要求の処理を拒否します。

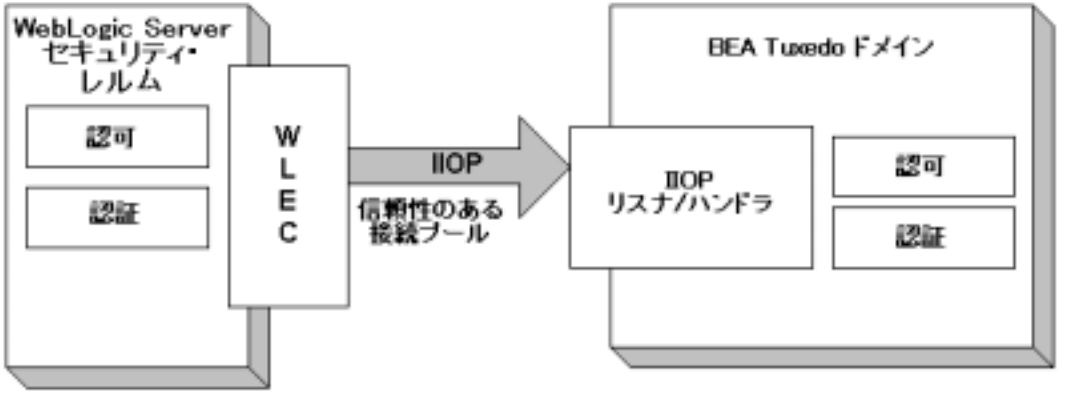
CORBA セキュリティ環境のシングル・サイン・オン

WebLogic Server のセキュリティ・レルムと BEA Tuxedo のドメインは、別々のスコープを持つセキュリティ定義と見なされます。それぞれは、ユーザおよびアクセス制御に関する独自のセキュリティ・データベースを持っています。ただし、WebLogic Enterprise Connectivity (WLEC) を使用すると、信頼性のある接続プールを構成する接続を介して、WebLogic Server セキュリティ・レルムで認証されたプリンシパルの ID を提示し、BEA Tuxedo ドメインで認証されたプリンシパルの ID を作成することができます。

注記　BEA Tuxedo 製品の CORBA セキュリティ環境のシングル・サイン・オン機能は一方向です。プリンシパルの ID は、WebLogic Server セキュリティ・レルムから BEA Tuxedo ドメインへの伝達のみ可能です。

図 3-2 は、CORBA セキュリティ環境におけるシングル・サイン・オンのしくみを示しています。

図 3-2CORBA セキュリティ環境のシングル・サイン・オン



シングル・サイン・オンを使用する場合、WebLogic Server ユーザの ID は、信頼性のある接続プールを構成するネットワーク接続を介して、BEA Tuxedo ドメイン内の CORBA オブジェクトに送信される IIOP 要求のサービス・コンテキストの一部として伝達されます。信頼性のある接続プール内の各ネットワーク接続は、定義されているプリンシパルの ID を使用して認証されています。信頼性のある接続プールを確立するには、パスワードおよび証明書を使用して認証します。

伝達された ID は、IIOP リスナ / ハンドラによって BEA Tuxedo ドメインでプリンシパル ID の代わりとして使用されます。代わりに使用される ID は、1 組のトークン (認証用と監査用にそれぞれ 1 つ) として表されます。これらのトークンは、BEA Tuxedo ドメインにあるターゲットの CORBA オブジェクトに伝達され、認証および監査で使用されます。

プリンシパル ID のマッピングを容易にするため、IIOP リスナ / ハンドラは認証プラグインを使用します。このプラグインは、プリンシパル ID を認証および監査トークンにマップします。これらのトークンは、ターゲットの CORBA オブジェクトに転送される要求の一部として伝達されます。ターゲットの CORBA オブジェクトはこれらのトークンを使用して、プリンシパルの ID など、要求のイニシエータについての情報を調べます。

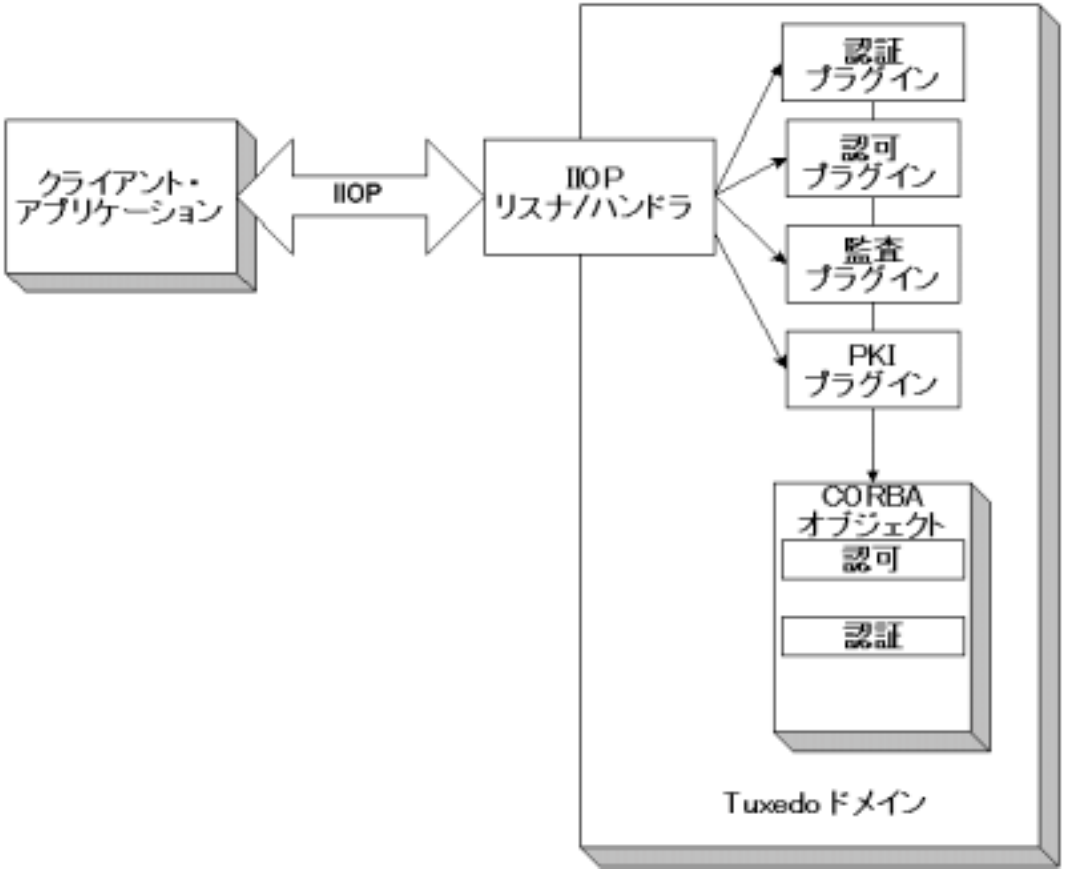
SSL プロトコルを使用すると、WebLogic Server レルムからの要求の機密性と整合性を保護できます。SSL 暗号化は、BEA Tuxedo ドメイン内の CORBA オブジェクトに対する IIOP 要求向けに用意されています。要求を保護するには、SSL プロトコルを使用するように WebLogic Connectivity および CORBA アプリケーションをコンフィギュレーションする必要があります。

シングル・サイン・オンの実装方法については、「第 8 章 シングル・サイン・オンのコンフィギュレーション」を参照してください。

BEA Tuxedo セキュリティ SPI

図 3-3 に示したように、BEA Tuxedo 製品で利用可能な認証、認可、監査、および公開鍵によるセキュリティ機能は、プラグイン・インターフェイスを通じて実装されます。このインターフェイスを使用することで、セキュリティ・プラグインを CORBA 環境に統合できるようになります。セキュリティ・プラグインは、特定のセキュリティ機能を実装するコード・モジュールです。

図 3-3BEA Tuxedo セキュリティ・サービス・プロバイダ・インターフェイスのアーキテクチャ



BEA Tuxedo 製品には、表 3-2 に示したセキュリティ・プラグイン用のインターフェイスが用意されています。

表 3-2 BEA Tuxedo のセキュリティ・プラグイン

プラグイン	説明
認証	通信するプロセスどうしがお互いの ID を証明し合うことです。
認可	認可の機能により、管理者は CORBA アプリケーションへのアクセスを制御できます。つまり、管理者は、認可機能を使用して、CORBA アプリケーションのリソースまたはサービスに対するアクセス権をプリンシパル (認証されたユーザ) に許可するかどうかを決定します。
監査	操作要求とその結果に関する情報を収集、格納、および配布する方法です。監査証跡の記録からは、CORBA のセキュリティ方針に違反するアクションを実行したプリンシパルや、そのようなアクションを実行しようとしたプリンシパルを判別できます。また、これらの記録から、試行された操作、失敗した操作、および成功した操作を判別することもできます。
公開鍵の初期化	公開鍵ソフトウェアが公開鍵および秘密鍵を開けるようにします。たとえば、ゲートウェイ・プロセスでは、メッセージを復号化してから転送するために、特定の秘密鍵へのアクセスが必要なこともあります。
鍵管理	公開鍵ソフトウェアが公開鍵および秘密鍵を管理および使用できるようにします。なお、メッセージ・ダイジェストとセッション・キーは、このインターフェイスを使用して暗号化および復号化されますが、公開鍵暗号を使用するバルク・データの暗号化は行われません。バルク・データの暗号化は、対称鍵暗号を使用して行われます。

表 3-2 BEA Tuxedo のセキュリティ・プラグイン (続き)

プラグイン	説明
証明ルックアップ	公開鍵ソフトウェアが、所定のプリンシパルに対する X.509v3 デジタル証明書を取得できるようにします。デジタル証明書は、Lightweight Directory Access Protocol (LDAP) など、適切な証明書リポジトリを使用して格納することができます。
証明解析	公開鍵ソフトウェアが、簡単なプリンシパル名と X.509v3 デジタル証明書を関連付けることができるようにします。パーサは、デジタル証明書を解析して、デジタル証明書に関連付けるプリンシパル名を生成します。
証明書の検証	公開鍵ソフトウェアが特定のビジネス・ロジックに基づいて X.509v3 デジタル証明書を検証することができます。
証明資料のマッピング	公開鍵ソフトウェアは、鍵を開くために必要な証明資料にアクセスしたり、認可トークンおよび監査トークンを提供したりすることができます。

SPI の仕様は、BEA 社と専用契約を結んだサード・パーティのセキュリティ・ベンダだけが利用できます。セキュリティ機能をカスタマイズする場合は、これらのセキュリティ・ベンダまたは BEA プロフェッショナル・サービスにお問い合わせください。たとえば、公開鍵によるセキュリティ機能をカスタマイズする場合は、適切なセキュリティ・プラグインを提供するサード・パーティのセキュリティ・ベンダまたは BEA プロフェッショナル・サービスに問い合わせる必要があります。

セキュリティ・プラグインの詳細 (インストール手順およびコンフィギュレーション手順を含む) については、BEA 社の営業担当者にお問い合わせください。

章

2 SSL 技術の概要

ここでは、次の内容について説明します。

- SSL プロトコル
- デジタル証明書
- 認証局
- 証明書のリポジトリ
- PKI (Public Key Infrastructure)
- PKCS-5 および PKCS-8 の準拠
- サポートされている公開鍵のアルゴリズム
- サポートされている対称鍵のアルゴリズム
- サポートされているメッセージ・ダイジェスト・アルゴリズム
- サポートされている暗号スイート
- デジタル証明書の規格

SSL プロトコル

Secure Sockets Layer (SSL) プロトコルを使用すると、以下の重要な機能を CORBA アプリケーションに統合できます。

- 機密性

機密性とは、意図した受信者以外の第三者に対して通信の内容を秘密にしておく機能のことです。このために、強力な暗号アルゴリズムを使用してデータを暗号化します。SSL プロトコルは、双方の通信当事者がサポートしている最強のアルゴリズムを調整し、データ暗号化用の鍵に関して合意するためのセキュリティ・メカニズムを提供します。

- 整合性

  整合性とは、データが転送中に変更されていないことを保証することです。双方がアルゴリズムと鍵について合意するためのハンドシェーク・メカニズムでは、受信したデータが変更されていた場合に検出できるように、SSL 接続の両端で共有データの整合性を秘密にすることもできます。

- 認証

  認証とは、通信相手を確認する機能のことです。CORBA クライアントおよびサーバ・アプリケーションは、デジタル証明書と公開鍵セキュリティ機能を使用することで互いを認証できます。これにより、双方は信頼性のある相手と通信していることを確認できます。SSL プロトコルでは、X.509 デジタル証明書を使用して BEA Tuxedo ドメインに対してプリンシパルを認証するためのメカニズムが提供されます。証明書による認証は、パスワードによる認証の代わりの方法として利用できます。

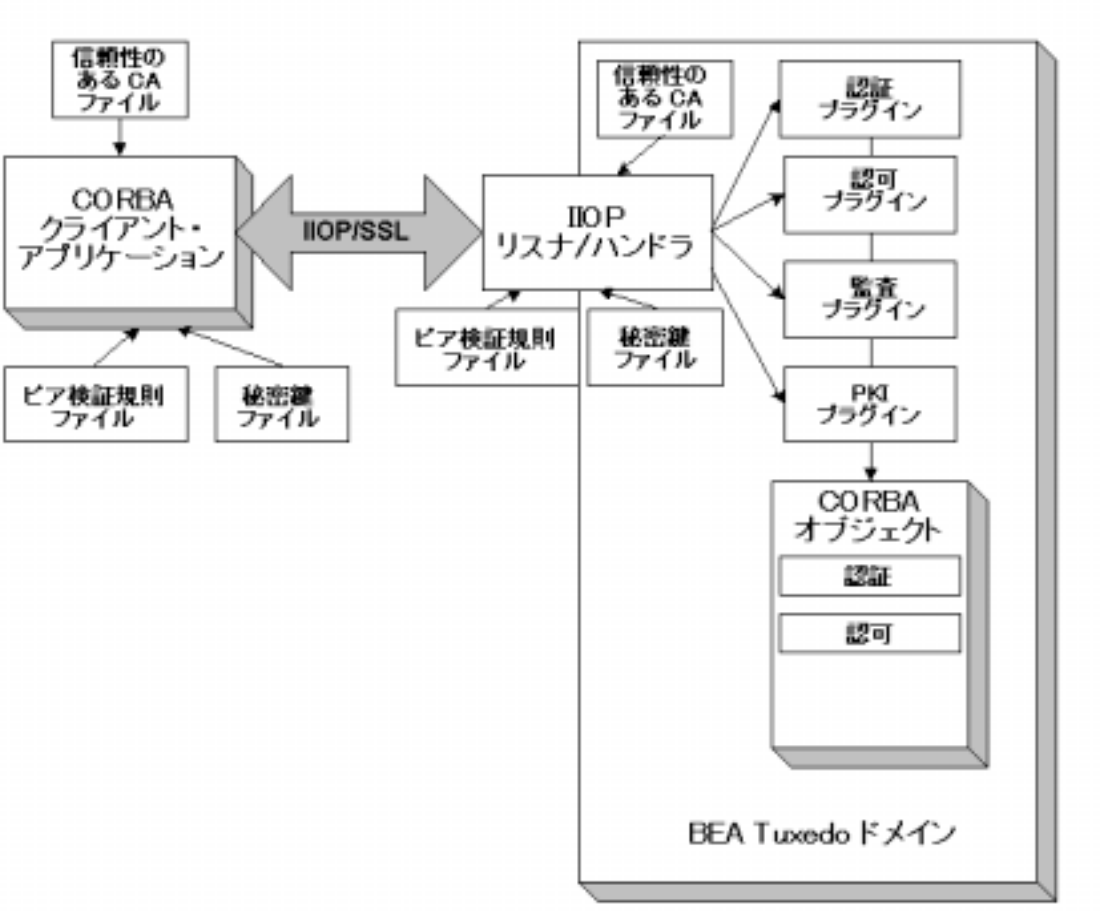
SSL プロトコルでは、2 つのアプリケーションがネットワーク接続を介して通信相手の ID を認証し、アプリケーション間でやり取りされるデータを暗号化することで、安全な接続を提供します。SSL プロトコルを使用する場合、ターゲットは常に自身をイニシエータに対して認証します。ターゲットから要求があった場合は、イニシエータが自身をターゲットに対して認証することもできます。暗号化によって、データは意図した受信者以外には理解できない形式でネットワーク上を転送されます。SSL 接続はハンドシェークで始まります。ハンドシェークでは、アプリケーションがデジタル証明書を交換し、使用する暗号アルゴリズムに関して合意し、以降のセッション用の暗号化キーを生成します。

SSL では、公開鍵暗号化を使用して認証します。公開鍵暗号化では、プリンシパルやその他のエンティティ (IIOP リスナ / ハンドラやアプリケーション・サーバ) に対して 1 組の非対称鍵が生成されます。公開鍵で暗号化したデータは対応する秘密鍵でのみ解読できるようになっています。逆に、秘密鍵で暗号化したデータは対応する公開鍵でのみ解読できます。所有者だけが

メッセージを解読できるように、公開鍵の保護には十分に注意を払います。しかし、公開鍵はその所有者宛のメッセージを暗号化できるように、自由に配布します。

図 3-4 は、CORBA セキュリティ環境における SSL プロトコルのしくみを示しています。

図 3-4CORBA セキュリティ環境の SSL プロトコル



CORBA セキュリティ環境で SSL プロトコルを使用する場合、IIOP リスナ / ハンドラは開始元プリンシパルに対して自身を認証します。IIOP リスナ / ハンドラは、デジタル証明書を開始元プリンシパルに提示します。SSL 接続を調整するには、クライアント・アプリケーションが IIOP リスナ / ハンドラを認証する必要がありますが、IIOP リスナ / ハンドラはすべてのクライアント・アプリケーションを SSL 接続に受け入れます。こうした認証は、サーバ認証と呼ばれます。

サーバ認証を使用する場合、開始元クライアント・アプリケーションは、信頼性のある認証局のデジタル証明書を取得する必要があります。IIOP リスナ / ハンドラは、ID を表す秘密鍵およびデジタル証明書を取得する必要があります。サーバ認証は、顧客が個人データを送信する前に安全な接続を確立する必要がある場合に、インターネットではよく用いられる方法です。この場合、クライアント・アプリケーションは Web ブラウザとほぼ同じ役割を持つことになります。

SSL バージョン 3.0 を使用すると、プリンシパルも IIOP リスナ／ハンドラに対して認証を行うことができます。こうした認証は、相互認証と呼ばれます。相互認証では、プリンシパルが IIOP リスナ / ハンドラに対してデジタル証明書を提示します。相互認証を使用すると、IIOP リスナ / ハンドラとプリンシパルの双方がそれぞれの ID を表す秘密鍵およびデジタル証明書を持つ必要があります。相互認証は、信頼性のあるプリンシパルだけにアクセスを制限する場合に便利な方法です。

SSL プロトコルとデジタル証明書を使用するインフラストラクチャは、BEA Tuxedo 製品のインストール・ディレクトリにライセンスをインストールすることで利用できるようになります。詳細については、『BEA Tuxedo システムのインストール』を参照してください。

デジタル証明書

デジタル証明書は、インターネットなどのネットワーク経由で、プリンシパルおよびエンティティを一意に識別するために使用される電子文書です。デジタル証明書は、信頼性のある第三者機関である「認証局」によって認定さ

れたプリンシパルまたはエンティティの ID を特定の公開鍵に安全な方法で結び付けます。公開鍵と秘密鍵の組み合わせにより、デジタル証明書の所有者に一意な ID が提供されます。

デジタル証明書は、特定の公開鍵が特定のプリンシパルまたはエンティティに属することを検証します。デジタル証明書の受信者は、デジタル証明書に記載されている公開鍵を使用して、その公開鍵に対応する秘密鍵でデジタル署名が作成されたことを検証します。検証が成功すると、デジタル証明書で指定されたサブジェクトが、対応する秘密鍵の所有者であること、および、そのサブジェクトによってデジタル署名が作成されたことを、一連の処理で確認できたことになります。

デジタル証明書には、次のようなさまざまな情報が含まれています。

- サブジェクト (所持者、所有者) の名前、およびサブジェクトを一意に識別するためのその他の ID 情報 (デジタル証明書を使用する Web サーバの URL、個人の電子メール・アドレスなど)
- サブジェクトの公開鍵
- デジタル証明書を発行した認証局の名前
- シリアル番号
- デジタル証明書の有効期間または存続期間 (開始日と終了日を定義)

最も広く知られているデジタル証明書の形式は、ITU-T X.509 国際規格によって定義された形式です。したがって、X.509 準拠のアプリケーションであれば、デジタル証明書の読み書きを行えます。CORBA セキュリティ環境の PKI は、X.509 バージョン 3 つまり X.509v3 に準拠したデジタル証明書を認識します。

認証局

デジタル証明書は、認証局によって発行されます。デジタル証明書と公開鍵の発行対象に対して ID を保証できる、信頼性のある第三者機関または企業は、認証局となることができます。デジタル証明書を作成する場合、認証局

は、改ざんがあった場合に検出できるように、秘密鍵を使用して署名します。認証局は次に、署名付きのデジタル証明書を要求元のサブジェクトに返します。

サブジェクトは、認証局の公開鍵を使用して、デジタル証明書が認証局によって発行されたことを検証できます。認証局は、下位レベルの認証局の公開鍵の有効性を証明する上位レベルの認証局のデジタル証明書を提供することで、公開鍵を利用可能にします。2 つ目の方法を用いると、認証局が階層化されます。この階層の最上位は、ルート鍵と呼ばれる自己署名付きのデジタル証明書です。

暗号化メッセージの受信者が、信頼する上位認証局によって署名されたデジタル証明書を持ち、この証明書に認証局の公開鍵が含まれている場合、認証局の秘密鍵の信頼性を再帰的に高めることができます。この意味で、デジタル証明書は、デジタル署名の信頼性を確認する足がかりとなります。つまり、最終的には、いくつかの上位認証局の公開鍵の信頼性を確認するだけで済みます。一連のデジタル証明書を確認することにより、多数のユーザのデジタル署名の信頼性を証明できます。

つまり、デジタル署名は通信エンティティの ID を証明しますが、署名の信頼度は、署名を検証するための公開鍵を信頼できる、というレベルと同じです。

証明書のリポジトリ

特定のサブジェクト用の公開鍵およびその ID を利用可能にし、検証に使用できるようにするため、デジタル証明書をリポジトリに公開したり、その他の方法で公開できます。証明書のリポジトリは、デジタル証明書およびその他の情報で構成されるデータベースであり、リポジトリ内の情報は、取得したり、デジタル署名を検証するために使用できます。情報を取得するには、必要に応じて直接リポジトリ内のデジタル証明書を要求することにより、自動的に行います。

CORBA セキュリティ環境では、Lightweight Directory Access Protocol (LDAP) が証明書のリポジトリとして使用されます。BEA 社では、特定の LDAP サーバを提供しておらず、また推奨もしていません。選択した LDAP サーバは、X.500 スキーマ定義と LDAP バージョン 2 または 3 プロトコルをサポートしている必要があります。

PKI (Public Key Infrastructure)

PKI (Public Key Infrastructure) は、公開鍵暗号のアプリケーションをサポートするプロトコル、サービス、および標準で構成されます。PKI は比較的新しい技術であるため、定義はあいまいです。たとえば、単に、公開鍵のデジタル証明書に基づいた、信頼性を示す階層構造を指す場合があります。また、別のコンテキストでは、エンド・ユーザ用アプリケーションのデジタル署名および暗号化サービスを意味する場合もあります。

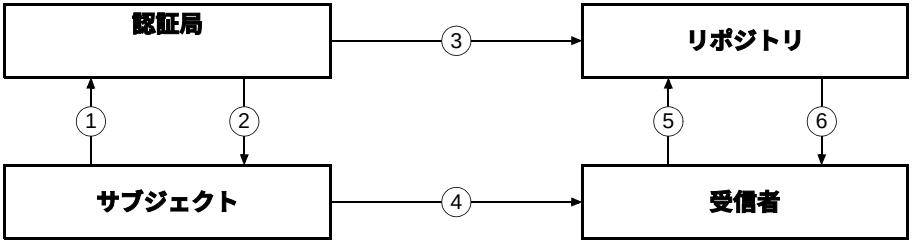
PKI を規定する単一の標準はありませんが、標準の策定を進める動きはあります。現時点では、標準が策定されるかどうか、またはさまざまな相互運用レベルの PKI が複数誕生するかどうかは不明です。この意味で、PKI 技術の現状は、インターネットによる広範囲の接続が可能になるまでの、1980 年代の LAN および WAN 技術に似ていると言えます。

PKI には、次のようなサービスがあります。

- キー登録。公開鍵の新しいデジタル証明書を発行します。
- 証明書の取り消し。発行済みのデジタル証明書および秘密鍵を取り消します。
- キー選択。パーティの公開鍵を取得します。
- 信頼性の評価。証明書の有効性、および証明書で認可される操作を決定します。

図 3-5 は、PKI の処理の流れを示します。

図 3-5PKI の処理の流れ



7. 認証局はサブジェクトの ID を確認して、デジタル証明書を発行します。
8. 認証局またはサブジェクトは、LDAP などの証明書のリポジトリにデジタル証明書を公開します。
9. サブジェクトは、秘密鍵で電子メッセージにデジタル署名して、送信者が認証済みであること、メッセージが完全であること、およびメッセージを否認できないことを確認します。その後、メッセージを受信者に送信します。
10. 受信者は、送信者の証明書を証明書のリポジトリから取り出し、その中から公開鍵を取り出します。

BEA Tuxedo 製品には、認証局になるためのツールは用意されていません。BEA 社では、VeriSign や Entrust などのサード・パーティの認証局を利用することをお勧めしています。BEA 社の公開鍵 SPI により、BEA Tuxedo のユーザは、PKI ソフトウェアのベンダを自由に選択し、PKI のセキュリティ・ソリューションを使用できます。詳細については、第 3 章の 25 ページ「シングル・サイン・オン」を参照してください。

PKCS-5 および PKCS-8 の準拠

RSA Laboratories を中心とする主要な通信会社のグループによって規定された、非公式でありながら業界では認知されている公開鍵ソフトウェアの標準があります。これを PKCS (Public-Key Cryptography Standards) と言います。BEA Tuxedo 製品では、PKCS-5 および PKCS-8 を使用して、SSL プロトコルで使用する秘密鍵を保護します。

- PKCS-5 は、DES を使用してデータを保護するパスワード・ベースの暗号化を使用するための形式の仕様です。
- PKCS-8 は、PKCS-5 を使用して秘密鍵を暗号化する機能を含めた、秘密鍵を保存する形式の仕様です。

サポートされている公開鍵のアルゴリズム

公開鍵 (または非対称鍵) のアルゴリズムは、次の 2 つの鍵の組み合わせによって実装されます。これらの鍵は異なりますが、数学的には関連性があります。

- 公開鍵。広範囲にわたって公開される鍵であり、デジタル署名を検証したり、データを理解できない形式に変換するために使用します。
- 秘密鍵。秘密に扱われる鍵であり、デジタル署名を作成したり、データを元の形式に戻すために使用します。

CORBA セキュリティ環境の公開鍵セキュリティ機能は、デジタル証明書のアルゴリズムもサポートしています。デジタル署名のアルゴリズムは、単にデジタル署名を実現するための公開鍵アルゴリズムです。

BEA Tuxedo 製品は、RSA (Rivest、Shamir、Adelman の頭文字を取ったもの) アルゴリズム、Diffie-Hellman アルゴリズム、および Digital Signature Algorithm (DSA) をサポートしています。DSA 以外のデジタル証明書のアルゴリズムは、デジタル署名と暗号化に使用できます。DSA は、デジタル署名には使用できますが、暗号化には使用できません。

サポートされている対称鍵のアルゴリズム

対称鍵アルゴリズムでは、同じ鍵を使用して、メッセージの暗号化と復号化を行います。公開鍵による暗号化システムでは、通信し合う 2 つのエンティティ間で送信されるメッセージの暗号化に対称鍵暗号化を使用します。対称鍵暗号は、公開鍵暗号より、少なくとも 1000 倍速く実行されます。

ブロック暗号は、対称鍵アルゴリズムの一種であり、固定長の平文 (暗号化されていないテキスト) のブロックを、同じ長さの暗号文 (暗号化されたテキスト) のブロックに変換します。この変換は、ランダムに生成されたセッション・キーの値に基づいて行われます。固定長は、ブロック・サイズと呼ばれます。

CORBA セキュリティ環境の公開鍵セキュリティ機能は、次の対称鍵アルゴリズムをサポートしています。

- DES-CBC (Data Encryption Standard for Cipher Block Chaining)

  DES-CBC は、CBC (Cipher Block Chaining) モードで実行する、64 ビット単位のブロック暗号です。DES-CBC の暗号化キーの長さは 56 ビット (64 ビットの暗号化キーから 8 パリティ・ビットを引いたもの) です。

- 2 つの鍵による Triple-DES (Data Encryption Standard)

  2 つの鍵による Triple-DES は、EDE (Encrypt-Decrypt-Encrypt) モードで実行する 128 ビットのブロック暗号です。これは、2 つの 56 ビットの暗号化キー (112 ビットの暗号化キー) を提供します。

最近では、Triple-DES を使用して、DES の暗号鍵を保護し、転送するのが一般的になりました。つまり、入力データ (この場合は単一の DES キー) は、暗号化、復号化、暗号化、の順に繰り返し処理されます (暗号化 / 復号化 / 暗号化のプロセス)。このうち、2 回行われる暗号化では、両方とも同じ暗号鍵が使用されます。

- RC2 (Rivest's Cipher 2)

  RC2 は、暗号鍵のサイズを変更できるブロック暗号です。

- RC4 (Rivest's Cipher 4)

  RC4 は、40 ～ 128 ビットの範囲で、暗号鍵のサイズを変更できるブロック暗号です。DES より高速であり、40 ビットの暗号鍵は輸出できます。米国籍の企業の海外子会社および海外支店であれば、56 ビットの暗号鍵を使用することができます。CORBA セキュリティ環境の公開鍵セキュリティ機能では暗号鍵の長さを 128 ビットに制限していますが、米国では実質的に、RC4 にはどんな長さの鍵を使用することもできます。

BEA Tuxedo 製品をお使いのお客様は、このアルゴリズムのリストを拡張したり変更することはできません。

サポートされているメッセージ・ダイジェスト・アルゴリズム

CORBA セキュリティ環境では、MD5 および SHA-1 (Secure Hash Algorithm 1) メッセージ・ダイジェスト・アルゴリズムをサポートしています。MD5 と SHA-1 は、どちらも有名な一方向のハッシュ・アルゴリズムです。一方向のハッシュ・アルゴリズムでは、メッセージが「メッセージ・ダイジェスト」または「ハッシュ値」と呼ばれる固定長の数値文字列に変換されます。

MD5 は、128 ビットのハッシュ値を生成する、高速のアルゴリズムです。32 ビットのマシンでの使用に適しています。SHA-1 は、160 ビットのハッシュ値を生成する、セキュリティ・レベルの高いアルゴリズムですが、処理速度は MD5 よりやや遅くなります。

サポートされている暗号スイート

暗号スイートとは、通信の整合性を保護するための鍵暗号アルゴリズム、対称暗号アルゴリズム、および Secure Hash Algorithm からなる SSL 暗号化方式のことです。たとえば、暗号スイート RSA_WITH_RC4_128_MD5 では、鍵暗号用に RSA、バルク暗号化用に 128 ビット鍵の RC4、メッセージ・ダイジェスト用に MD5 を使用します。

CORBA セキュリティ環境では、表 3-3 に示した暗号スイートをサポートしています。

表 3-3 CORBA セキュリティ環境でサポートされている SSL 暗号スイート

暗号スイート	鍵暗号の種類	対称鍵の強度
SSL_RSA_WITH_RC4_128_SHA	RSA	128
SSL_RSA_WITH_RC4_128_MD5	RSA	128
SSL_RSA_WITH_DES_CDC_SHA	RSA	56
SSL_RSA_EXPORT_WITH_RC4_40_MD5	RSA	40
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA	RSA	40
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	RSA	40
SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	Diffie-Hellman	40
SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	Diffie-Hellman	40
SSL_RSA_WITH_3DES_EDE_CBC_SHA	RSA	112
SSL_RSA_WITH_NULL_SHA	RSA	0
SSL_RSA_WITH_NULL_MD5	RSA	0

デジタル証明書の規格

CORBA セキュリティ環境では、X.509v3 規格に準拠したデジタル証明書をサポートしています。X.509v3 規格には、デジタル証明書の形式が指定されています。Verisign や Entrust などの認証局から証明書を取得することをお勧めします。

章

3 CORBA のセキュリティの基本概念

ここでは、以下の内容について説明します。

- リンク・レベルの暗号化
- パスワードによる認証
- SSL プロトコル
- 証明書による認証
- 認証プラグインの使い方
- 認可
- 監査
- シングル・サイン・オン
- PKI プラグイン
- CORBA のセキュリティ機能に関してよくある質問

リンク・レベルの暗号化

リンク・レベルの暗号化 (LLE) は、ネットワーク・リンク上で送受信されるメッセージのデータ機密性を保護します。LLE の目的は、BEA Tuxedo システムのメッセージや CORBA アプリケーションが生成したメッセージの内容を、ネットワーク上の侵入者に知られないようにすることです。また、対称鍵による暗号化技術である RC4 を使用します。RC4 では、暗号化と復号化の際に同じ鍵を使用します。

LLE を使用する場合、BEA Tuxedo システムは、データを暗号化してからネットワーク・リンク上に送信し、データがネットワーク・リンクを離れると復号化します。システムは、データが通過するすべてのリンクで、この暗号化 / 復号化プロセスを繰り返します。したがって、LLE はポイント・ツー・ポイント機能と呼ばれます。

LLE を使用すると、CORBA アプリケーション内のマシンとドメイン間の通信を暗号化できます。

注記　LLE は、リモート CORBA クライアント・アプリケーションと IIOP リスナ / ハンドラ間の接続の保護には使用できません。

LLE セキュリティには、0 ビット (暗号化なし)、56 ビット (国際版)、128 ビット (米国およびカナダ版) の 3 つのレベルがあります。国際版の LLE では、0 ビットと 56 ビットの暗号化が可能です。米国およびカナダ版の LLE では、0 ビット、56 ビット、および 128 ビットの暗号化が可能です。

LLE のしくみ

LLE は次のように機能します。

1. システム管理者が、暗号化レベルを制御するために LLE を使用するプロセスのパラメータを設定します。
   - 1 つ目は、プロセスで受け付ける暗号化の最低レベルを設定するコンフィギュレーション・パラメータです。これは、0、56、128 のいずれかのキー長で指定します。

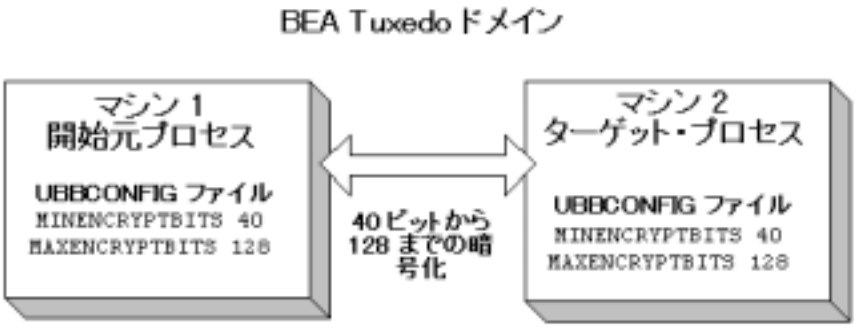
- 2 つ目は、プロセスでサポートされる暗号化の最高レベルを設定するコンフィギュレーション・パラメータです。これも、0、56、128 のいずれかのキー長で指定します。

便宜上、上記の 2 つのパラメータを (min, max) の形式で表記します。たとえば、あるプロセスの値が (56, 128) の場合は、56 ビットから 128 ビットまでの暗号化がサポートされることを示します。

2. イニシエータ・プロセスが通信セッションを開始します。
3. ターゲット・プロセスは初期接続を受け取り、通信する 2 つのプロセスが使用する暗号化レベルを調整します。
4. 2 つのプロセスは、双方がサポートするキーの最大サイズについて合意します。
5. コンフィギュレーションした最大キー・サイズは、インストールされているソフトウェアの機能に合わせて縮小されます。この手順はリンクの調整時に行う必要があります。コンフィギュレーション時には、特定のマシンにインストールされている暗号化パッケージを確認できないからです。
6. プロセスは、調整済みの暗号化レベルでメッセージをやり取りします。

図 3-1 は、これまでの手順を示しています。

図 3-1LLE のしくみ



暗号化キー・サイズの調整

ネットワーク・リンクの両端にある 2 つのプロセスが通信し合うには、まず、これらのプロセスの暗号化キーのサイズが合っていなければなりません。これは、次の 2 段階の手順で確認されます。

1. 各プロセスで、それぞれの *min-max* 値が確認されます。
2. 両方のプロセスでサポートされる、キーの最大サイズが算出されます。

min-max 値の決定

2 つのプロセスのうち、どちらかが起動すると、BEA Tuxedo システムは、(1) `lic.txt` ファイルの LLE 情報を参照して、インストール済みの LLE のバージョンのビット暗号化機能をチェックし、(2) 両方のプロセスのコンフィギュレーション・ファイルで指定されている、特定の種類のリンクでの LLE の *min-max* 値をチェックします。続いて、BEA Tuxedo システムは次の処理を実行します。

- コンフィギュレーションされている *min-max* 値が、インストール済みの LLE のバージョンと対応する場合、ローカル・ソフトウェアは、この値をプロセスの *min-max* 値として割り当てます。
- コンフィギュレーションされている *min-max* 値が、インストール済みの LLE のバージョンと対応しない場合、ローカル・ソフトウェアからはエラーが返されます。たとえば、国際版の LLE がインストールされているときに、*min-max* 値が (0, 128) である場合です。この時点では、リンク・レベルの暗号化は不可能です。
- 特定の種類のリンクのコンフィギュレーションで、*min-max* 値が指定されていない場合、ローカル・ソフトウェアは、最小値として 0 を割り当て、最大値としてインストール済みの LLE のバージョンに対して可能な最高ビットの暗号化レートを割り当てます。つまり、米国およびカナダ版の LLE では (0, 128) を割り当てます。

共通のキー・サイズの検索

2 つのプロセスの *min-max* 値が決まると、キー・サイズの調整が開始します。この調整プロセスを暗号化したり、見えないようにする必要はありません。調整されたキー・サイズは、ネットワーク接続が確立されている間は有効です。

表 3-1 は、2 つのプロセス間で可能な *min-max* 値のすべての組み合わせを調整した結果のキー・サイズを示しています。上段のヘッダ行は、2 つのプロセスのうち、片方のプロセスの *min-max* 値を示しています。左側の列は、もう一方のプロセスの *min-max* 値を示しています。

表 3-1 プロセス間の調整結果

	(0, 0)	(0, 56)	(0, 128)	(56, 56)	(56, 128)	(128, 128)
(0, 0)	0	0	0	エラー	エラー	エラー
(0, 56)	0	56	56	56	56	エラー
(0, 128)	0	56	128	56	128	128
(56, 56)	エラー	56	56	56	56	エラー
(56, 128)	エラー	56	128	56	128	128
(128, 128)	エラー	エラー	128	エラー	128	128

初期化時の WSL および WSH の接続タイムアウト

ワークステーション・クライアントが初期化に費やせる時間は制限されています。デフォルトでは、LLE を使用しないアプリケーションでは 30 秒、LLE を使用するアプリケーションでは 60 秒に制限されています。この 60 秒には、暗号化されたリンクを調整する時間も含まれます。ただし、LLE がコンフィギュレーションされている場合は、`UBBCONFIG` ファイルの `MAXINITTIME` パラメータでワークステーション・リスナ (WSL) のサーバの値を変更するか、または `WS_MIB`(5) にある `T_WSL` クラスの `TA_MAXINITTIME` 属性の値を変更することによって、制限を変更できます。

開発プロセス

CORBA アプリケーションで LLE を使用するには、LLE を有効にするライセンスをインストールする必要があります。ライセンスのインストール方法については、『BEA Tuxedo システムのインストール』を参照してください。

LLE のインプリメンテーションは管理タスクの 1 つです。各 CORBA アプリケーションのシステム管理者は、暗号化のレベルを制御する *min-max* 値を UBBCONFIG ファイルで設定します。2 つの CORBA アプリケーションは、通信を確立すると、メッセージのやり取りに使用する暗号化のレベルを調整します。調整されたキー・サイズは、ネットワーク接続が確立されている間は有効です。

パスワードによる認証

CORBA セキュリティ環境では、PKI (Public Key Infrastructure) を全面的にデプロイする準備が整っていない既存の CORBA アプリケーションおよび新しい CORBA アプリケーションに対して認証機能を提供するためのパスワード・メカニズムをサポートしています。パスワードによる認証を使用すると、CORBA オブジェクトを呼び出すアプリケーションは、定義されているユーザ名およびパスワードを提示して、BEA Tuxedo ドメインに対して自身を認証します。

利用できるパスワード認証レベルは以下のとおりです。

- なし ― CORBA アプリケーションでは、パスワードまたはアクセスのチェックを行いません。
- アプリケーション・パスワード ― CORBA アプリケーションにアクセスするには、ユーザはドメイン・パスワードを入力する必要があります。
- ユーザ認証 ― CORBA アプリケーションにアクセスするには、ユーザはドメイン・パスワードだけでなくアプリケーション・パスワードも入力する必要があります。

- ACL － CORBA アプリケーションで認可を使用し、インターフェイス、キュー名、およびイベント名に対するアクセス制御のチェックを行います。ユーザに ACL が関連付けられていなければ、アクセス権が付与されていると見なされます。
- 必須の ACL － CORBA アプリケーションで認可を使用し、インターフェイス、キュー名、およびイベント名に対するアクセス制御のチェックを行います。必須の ACL の値は ACL と似ていますが、ユーザに ACL が関連付けられていない場合にはアクセスは拒否されます。

パスワードによる認証を使用する場合、クライアント・アプリケーションで `Tobj::PrincipalAuthenticator::logon()` メソッドを使用する方法と、`SecurityLevel2::PrincipalAuthenticator::authenticate()` メソッドを使用する方法があります。

パスワードによる認証を使用する場合、SSL プロトコルによって、アプリケーション間の通信の機密性と整合性を保護できます。詳細については、第 3 章の -11 ページ「SSL プロトコル」を参照してください。

パスワードによる認証のしくみ

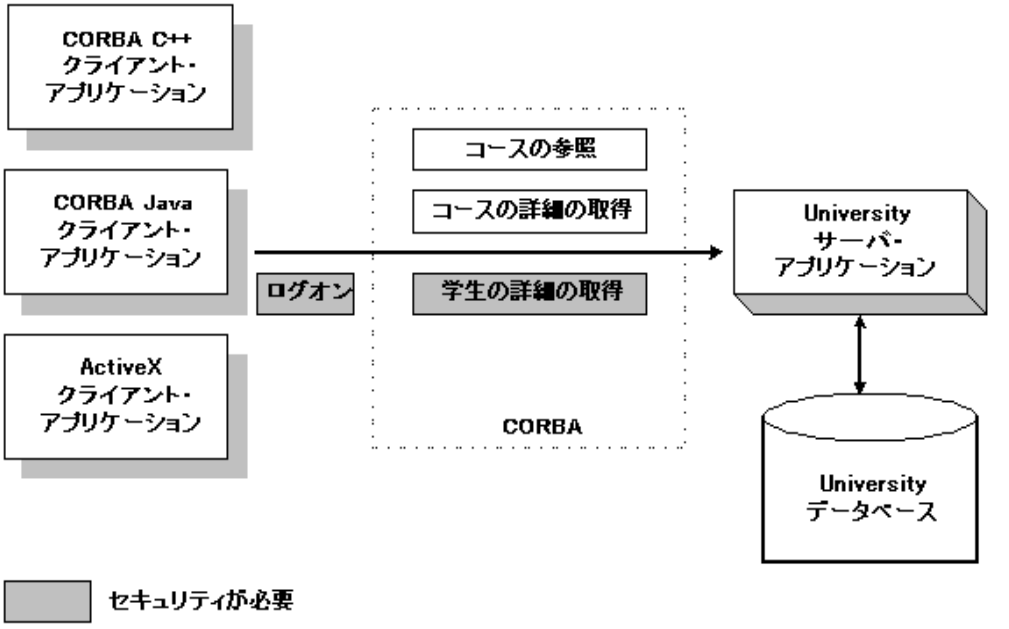
パスワードによる認証は次のように機能します。

1. 開始元アプリケーションは、次のいずれかの方法で BEA Tuxedo ドメインにアクセスします。
   - CORBA インターオペラブル・ネーミング・サービス (INS) のブートストラップ処理メカニズムを使用します。このメカニズムを使用するのは、別のベンダの クライアント ORB を使用する場合です。CORBA INS の使い方については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA プログラミング・リファレンス』を参照してください。
   - BEA のブートストラップ処理メカニズムを使用します。このメカニズムを使用するのは、BEA CORBA クライアント・アプリケーションを使用する場合です。

2. 開始元アプリケーションは、ユーザのクリデンシャルを取得します。開始元アプリケーションは、BEA Tuxedo ドメインが使用する証明資料を提示して、ユーザを認証する必要があります。この証明資料は、ユーザ名とパスワードで構成されています。
   - 開始元アプリケーションは、`PrincipalAuthenticator` オブジェクトを使用してセキュリティ・コンテキストを作成します。認証要求は、IIOP リスナ / ハンドラに送信されます。認証要求に含まれた証明資料は、提示された情報を検証する認証サーバまで安全に転送されます。
   - 検証が成功すると、BEA Tuxedo システムは、以降の呼び出しで使用される `Credentials` オブジェクトを作成します。ユーザの `Credentials` オブジェクトは、セキュリティ・コンテキストを表す `Current` オブジェクトに関連付けられます。
3. 開始元アプリケーションは、オブジェクト・リファレンスを使用して BEA Tuxedo ドメイン内の CORBA オブジェクトを呼び出します。要求は IIOP 要求にパッケージ化され、すでに確立されているセキュリティ・コンテキストとの関連付けを行う IIOP リスナ / ハンドラに転送されます。
4. IIOP リスナ / ハンドラは、開始元アプリケーションから要求を受信します。
5. IIOP リスナ / ハンドラは、要求を開始元アプリケーションのクリデンシャルと一緒に、適切な CORBA オブジェクトに転送します。

図 3-2 は、これまでの手順を示しています。

図 3-2 パスワードによる認証のしくみ



パスワードによる認証用の開発プロセス

パスワードによる認証を CORBA アプリケーションに定義するには、管理手順とプログラミング手順を実行する必要があります。表 3-2 と 表 3-3 には、それぞれの手順を示してあります。パスワードによる認証の管理手順の詳細については、第 7 章の 1 ページ「認証のコンフィギュレーション」を参照してください。プログラミング手順の詳細については、第 10 章の 1 ページ「セキュリティをインプリメントする CORBA アプリケーション」を参照してください。

表 3-2 パスワードによる認証の管理手順

手順	説明
1	UBBCONFIG ファイルの SECURITY パラメータを、APP_PW、USER_AUTH、ACL、または MANDATORY_ACL に設定します。
2	SECURITY パラメータを USER_AUTH、ACL、または MANDATORY_ACL にコンフィギュレーションした場合は、認証サーバ (AUTHSRV) を UBBCONFIG ファイルでコンフィギュレーションします。
3	tpusradd および tpgrpadd コマンドを使用して、許可されたユーザおよびグループ (IIOP リスナ / ハンドラを含む) のリストを定義します。
4	tmloadcf コマンドを使用して、UBBCONFIG ファイルをロードします。UBBCONFIG ファイルがロードされると、システム管理者はパスワードの入力を求められます。ここで入力するパスワードが CORBA アプリケーションのパスワードになります。

表 3-3 パスワードによる認証のプログラミング手順

手順	説明
1	Bootstrap オブジェクトを使用して BEA Tuxedo ドメインの SecurityCurrent オブジェクトへのリファレンスを取得するコード、または CORBA INS を使用して PrincipalAuthenticator オブジェクトへのリファレンスを取得するコードを記述します。
2	SecurityCurrent オブジェクトから PrincipalAuthenticator オブジェクトを取得するアプリケーション・コードを記述します。
3	Tobj::PrincipalAuthenticator::logon() または SecurityLevel2::PrincipalAuthenticator::authenticate() オペレーションを使用して、BEA Tuxedo ドメインのセキュリティ・コンテキストを取得するアプリケーション・コードを記述します。
4	UBBCONFIG ファイルがロードされたときに定義したパスワードの入力をユーザに要求するアプリケーション・コードを記述します。

SSL プロトコル

BEA Tuxedo 製品では、業界標準の SSL プロトコルを使用して、クライアント・アプリケーションとサーバ・アプリケーション間で安全な通信を確立します。SSL プロトコルを使用する場合、プリンシパルはデジタル証明書を使用して、自身の ID をピアに対して証明します。

CORBA セキュリティ環境の SSL プロトコルのデフォルト動作では、IIOP リスナ / ハンドラが自身の ID を、デジタル証明書を使用して SSL 接続を開始したプリンシパルに対して証明します。デジタル証明書は、各デジタル証明書が改ざんされていないこと、また有効期限が切れていないことを確認されます。一連の処理でデジタル証明書に問題があった場合、SSL 接続は終了します。また、デジタル証明書の発行者は信頼性のある認証局のリストと照合され、IIOP リスナ / ハンドラから受信したデジタル証明書が BEA Tuxedo ドメインの信頼する認証局が署名したものであるかどうか確認されます。

LLE と同じく、SSL プロトコルをパスワードによる認証で使用すると、クライアント・アプリケーションと BEA Tuxedo ドメイン間の通信の機密性と整合性を保護できます。SSL プロトコルをパスワードによる認証で使用する場合、`tmloadcf` コマンドを入力したときに `SEC_PRINCIPAL_NAME` パラメータで定義した IIOP リスナ / ハンドラのパスワードの入力を求められます。

SSL プロトコルのしくみ

SSL プロトコルは次のように機能します。

1. IIOP リスナ / ハンドラは、デジタル証明書を開始元アプリケーションに提示します。
2. 開始元アプリケーションは、信頼性のある認証局のリストと IIOP リスナ / ハンドラのデジタル証明書を照合します。
3. 開始元アプリケーションが IIOP リスナ / ハンドラのデジタル証明書を検証すると、アプリケーションと IIOP リスナ / ハンドラ間で SSL 接続が確立されます。

開始元アプリケーションは、パスワードまたは証明書による認証を使用して、自身を BEA Tuxedo ドメインに対して認証します。

図 3-3 は、SSL プロトコルのしくみを示しています。

図 3-3CORBA アプリケーションでの SSL プロトコルのしくみ

SSL プロトコルを使用するための要件

CORBA アプリケーションで SSL プロトコルを使用するには、SSL プロトコルおよび PKI を有効にするライセンスをインストールする必要があります。セキュリティ機能のライセンスのインストール方法については、『BEA Tuxedo システムのインストール』を参照してください。

SSL プロトコルのインプリメンテーションは柔軟性が高いので、ほとんどの PKI に対応します。BEA Tuxedo 製品では、デジタル証明書を LDAP 対応のディレクトリに保存する必要があります。LDAP 対応であれば、どのディレクトリ・サービスでもかまいません。また、CORBA アプリケーションで使用するデジタル証明書および秘密鍵の取得先の認証局を選択する必要があります。LDAP 対応のディレクトリ・サービスおよび認証局を準備してからでないと、SSL プロトコルを CORBA アプリケーションで使用できません。

SSL プロトコル用の開発プロセス

CORBA アプリケーションで SSL プロトコルを使用するための準備は、主に管理プロセスです。表 3-5 は、SSL プロトコルを使用できるようにインフラストラクチャを設定し、SSL プロトコルに合わせて IIOP リスナ / ハンドラをコンフィギュレーションするための管理手順の一覧です。管理手順の詳細については、第 4 章の 1 ページ「公開鍵によるセキュリティ機能の管理」および第 6 章の 1 ページ「SSL プロトコルのコンフィギュレーション」を参照してください。

管理手順を実行したら、パスワードによる認証と証明書による認証のどちらも CORBA アプリケーションで使用できます。詳細については、第 10 章の 1 ページ「セキュリティをインプリメントする CORBA アプリケーション」を参照してください。

注記　BEA CORBA C++ ORB をサーバ・アプリケーションとして使用している場合、ORB でも SSL プロトコルを使用するようにコンフィギュレーションできます。詳細については、第 6 章の 1 ページ「SSL プロトコルのコンフィギュレーション」を参照してください。

表 3-4 SSL プロトコル用の管理手順

手順	説明
1	LDAP 対応ディレクトリ・サービスをコンフィギュレーションします。BEA Tuxedo 製品のインストール時に、LDAP サーバの名前を入力する必要があります。
2	SSL プロトコルを使用するためのライセンスをインストールします。
3	IIOP リスナ / ハンドラのデジタル証明書および秘密鍵を認証局から取得します。
4	IIOP リスナ / ハンドラと認証局のデジタル証明書を LDAP 対応ディレクトリ・サービスで公開します。
5	ISL サーバ・プロセスの `SEC_PRINCIPAL_NAME`、`SEC_PRINCIPAL_LOCATION`、および `SEC_PRINCIPAL_PASSVAR` パラメータを `UBBCONFIG` ファイルで定義します。

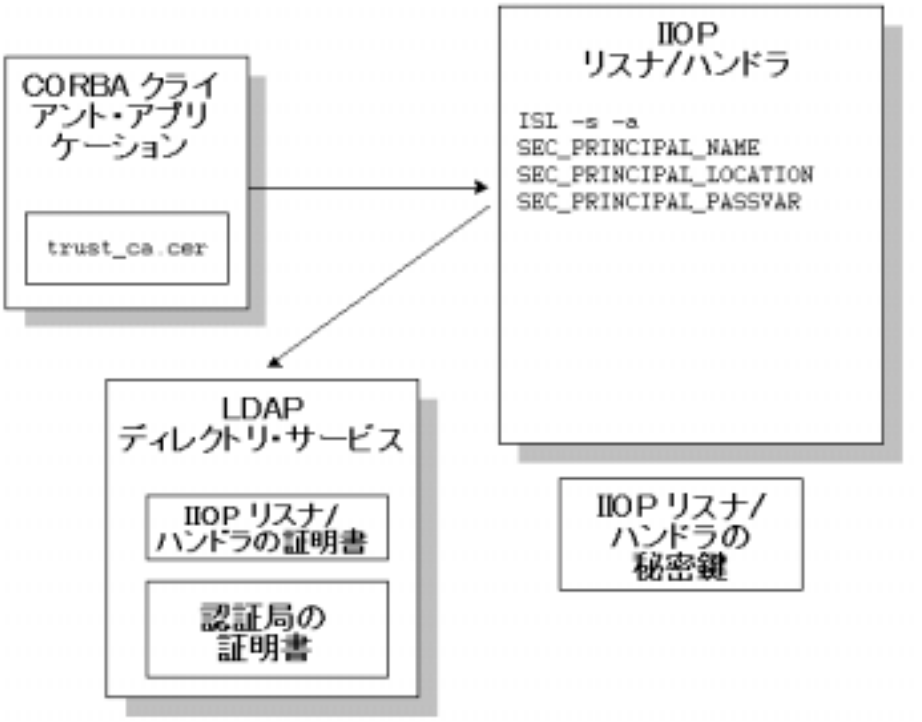
表 3-4 SSL プロトコル用の管理手順 (続き)

手順	説明
6	UBBCONFIG ファイルの SECURITY パラメータを NONE に設定します。
7	ISL コマンドの -S オプションを使用して、IIOP リスナ / ハンドラの安全な通信用のポートを定義します。
8	IIOP リスナ / ハンドラが信頼する認証局を定義する信頼性のある認証局ファイル (trust_ca.cer) を作成します。
9	tmloadcf コマンドを使用して、UBBCONFIG ファイルをロードします。
10	オプションで、IIOP リスナ / ハンドラのピア規則ファイル (peer_val.rul) を作成します。
11	オプションで、社内のディレクトリ階層に合わせて LDAP 検索フィルタ・ファイルを変更します。

SSL プロトコルをパスワードによる認証で使用する場合、UBBCONFIG ファイルの SECURITY パラメータを目的の認証レベルに設定し、必要であれば、認証サーバ (AUTHSRV) をコンフィギュレーションします。パスワードによる認証の管理手順の詳細については、第 3 章の -6 ページ「パスワードによる認証」を参照してください。

図 3-4 は、SSL プロトコルを使用する場合の CORBA アプリケーションのコンフィギュレーションを示しています。

図 3-4CORBA アプリケーションで SSL プロトコルを使用するコンフィギュレーション



証明書による認証

証明書による認証では、SSL 接続の両端がそれぞれの ID を互いに証明する必要があります。CORBA セキュリティ環境では、IIOP リスナ / ハンドラは自身のデジタル証明書を、SSL 接続を開始したプリンシパルに対して提示します。イニシエータは、IIOP リスナ / ハンドラが使用する一連のデジタル証明書を提示して、イニシエータの ID を証明します。

一連のデジタル証明書の検証に成功すると、IIOP リスナ / ハンドラはデジタル証明書のサブジェクトから識別名の値を取り出します。CORBA セキュリティ環境では、サブジェクトの識別名の電子メール・アドレス要素をプリンシパルの ID として使用します。IIOP リスナ / ハンドラは、プリンシパルの代わりにプリンシパルの ID を使用して、開始元アプリケーションと BEA Tuxedo ドメイン間のセキュリティ・コンテキストを確立します。

プリンシパルが認証されたら、要求を開始したプリンシパルと IIOP リスナ / ハンドラは、双方がサポートしている暗号化の種類とレベルを表す暗号スイートについて合意します。また、暗号鍵についても合意し、以降のすべてのメッセージについて同時に暗号化を開始します。

図 3-5 は、証明書による認証の概念を簡単にまとめたものです。

図 3-5 証明書による認証

証明書による認証のしくみ

証明書による認証は次のように機能します。

1. 開始元アプリケーションは、次のいずれかの方法で BEA Tuxedo ドメインにアクセスします。

- CORBA INS のブートストラップ処理メカニズムを使用します。このメカニズムを使用するのは、別のベンダの クライアント ORB を使用する場合です。CORBA INS の使い方については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA プログラミング・リファレンス』を参照してください。
- BEA のブートストラップ処理メカニズムを使用します。このメカニズムを使用するのは、BEA クライアント ORB を使用する場合です。

2. 開始元アプリケーションは、URL (`corbaloc://host:port` または `corbalocs://host:port` の形式) を使用して Bootstrap オブジェクトをインスタンス化し、`SecurityLevel2::PrincipalAuthenticator::authenticate` オペレーションの結果として返される `SecurityLevel2::Credentials` オブジェクトの属性を設定して、保護の要件を制御します。

注記　`SecurityLevel2::Current::authenticate()` メソッドを使用しても、ブートストラップ処理のセキュリティを確保し、証明書による認証を使用するように指定できます。

3. 開始元アプリケーションは、プリンシパルのデジタル証明書および秘密鍵を取得します。この情報を取得するには、プリンシパルの秘密鍵および証明書にアクセスするための証明資料を提示しなければならない場合があります。通常、証明資料はパスワードではなくパス・フレーズです。

   `SecurityLevel2::PrincipalAuthenticator::authenticate()` メソッドの結果として、セキュリティ・コンテキストが確立されます。

   IIOP リスナ / ハンドラは、認証プロセスの一部として、アプリケーションのデジタル証明書を受信して検証します。

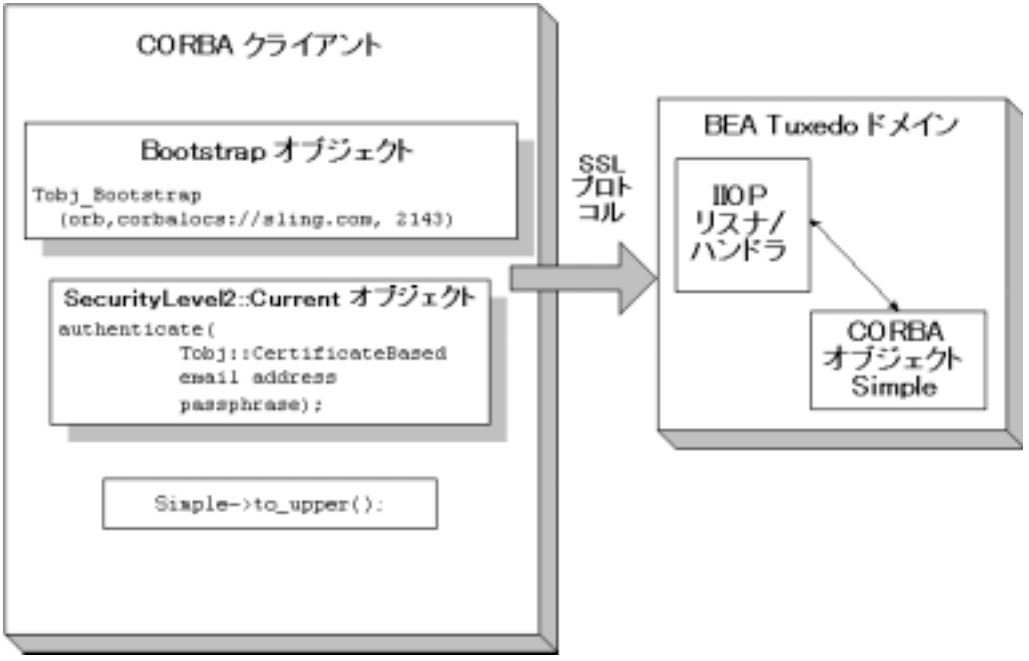
4. 検証が成功すると、BEA Tuxedo システムは `Credentials` オブジェクトを作成します。プリンシパルの `Credentials` オブジェクトは、現在の実行スレッドのセキュリティ・コンテキストを表します。

5. 開始元アプリケーションは、オブジェクト・リファレンスを使用して BEA Tuxedo ドメイン内の CORBA オブジェクトを呼び出します。

6. 要求は IIOP 要求にパッケージ化され、すでに確立されているセキュリティ・コンテキストとの関連付けを行う IIOP リスナ / ハンドラに転送されます。

7. 要求はデジタル署名され、暗号化されてから、IIOP リスナ / ハンドラに送信されます。BEA Tuxedo システムは、要求の署名と暗号化を実行します。
8. IIOP リスナ / ハンドラは、開始元アプリケーションから要求を受信します。要求が解読されます。
9. IIOP リスナ / ハンドラは、プリンシパルの subjectDN の電子メール・コンポーネントを受信し、ユーザの ID として使用します。
10. IIOP リスナ / ハンドラは、要求をプリンシパルの関連トークンと一緒に、適切な CORBA オブジェクトに転送します。

図 3-6 証明書による認証のしくみ



証明書による認証用の開発プロセス

CORBA アプリケーションで証明書による認証を使用するには、SSL プロトコルおよび PKI を有効にするライセンスをインストールする必要があります。ライセンスのインストール方法については、『BEA Tuxedo システムのインストール』を参照してください。

証明書による認証を CORBA アプリケーションで使用するには、管理手順とプログラミング手順を実行する必要があります。表 3-5 と 表 3-6 には、それぞれの手順を示してあります。管理手順の詳細については、第 4 章の 1 ページ「公開鍵によるセキュリティ機能の管理」および第 6 章の 1 ページ「SSL プロトコルのコンフィギュレーション」を参照してください。

表 3-5 証明書による認証の管理手順

手順	説明
1	LDAP 対応ディレクトリ・サービスをコンフィギュレーションします。BEA Tuxedo 製品のインストール時に、LDAP サーバの名前を入力する必要があります。
2	SSL プロトコルを使用するためのライセンスをインストールします。
3	IIOP リスナ / ハンドラのデジタル証明書および秘密鍵を認証局から取得します。
4	CORBA クライアント・アプリケーションのデジタル証明書および秘密鍵を認証局から取得します。
5	CORBA クライアント・アプリケーションおよび IIOP リスナ / ハンドラの秘密鍵ファイルをユーザのホーム・ディレクトリまたは `$TUXDIR\udataobj\security\keys` に格納します。
6	IIOP リスナ / ハンドラ、CORBA アプリケーション、および認証局のデジタル証明書を LDAP 対応ディレクトリ・サービスで公開します。
7	ISL サーバ・プロセスの `SEC_PRINCIPAL_NAME`、`SEC_PRINCIPAL_LOCATION`、および `SEC_PRINCIPAL_PASSVAR` を `UBBCONFIG` ファイルで定義します。

表 3-5 証明書による認証の管理手順 (続き)

手順	説明
8	UBBCONFIG ファイルの SECURITY パラメータを USER_AUTH、ACL、または MANDATORY_ACL に設定します。
9	認証サーバ (AUTHSRV) を UBBCONFIG ファイルでコンフィギュレーションします。
10	tpusradd および tpgrpadd コマンドを使用して、CORBA アプリケーションの許可されたユーザおよびグループを定義します。
11	ISL コマンドの -S オプションを使用して、IIOP リスナ / ハンドラの SSL 通信用のポートを定義します。
12	ISL コマンドの -a オプションを使用して、IIOP リスナ / ハンドラで証明書による認証を有効にします。
13	IIOP リスナ / ハンドラが信頼する認証局を定義する信頼性のある認証局ファイル (trust_ca.cer) を作成します。
12	CORBA クライアント・アプリケーションが信頼する認証局を定義する信頼性のある認証局ファイル (trust_ca.cer) を作成します。
13	tmloadcf コマンドを使用して、UBBCONFIG ファイルをロードします。SEC_PRINCIPAL_NAME パラメータで定義されている IIOP リスナ / ハンドラのパスワードの入力を求められます。
14	オプションで、CORBA クライアント・アプリケーションおよび IIOP リスナ / ハンドラのピア規則ファイル (peer_val.rul) を作成します。
15	オプションで、社内のディレクトリ階層に合わせて LDAP 検索フィルタ・ファイルを変更します。

図 3-7 は、証明書による認証を使用する場合の CORBA アプリケーションのコンフィギュレーションを示しています。

図 3-7CORBA アプリケーションで証明書による認証を使用するコンフィギュレーション

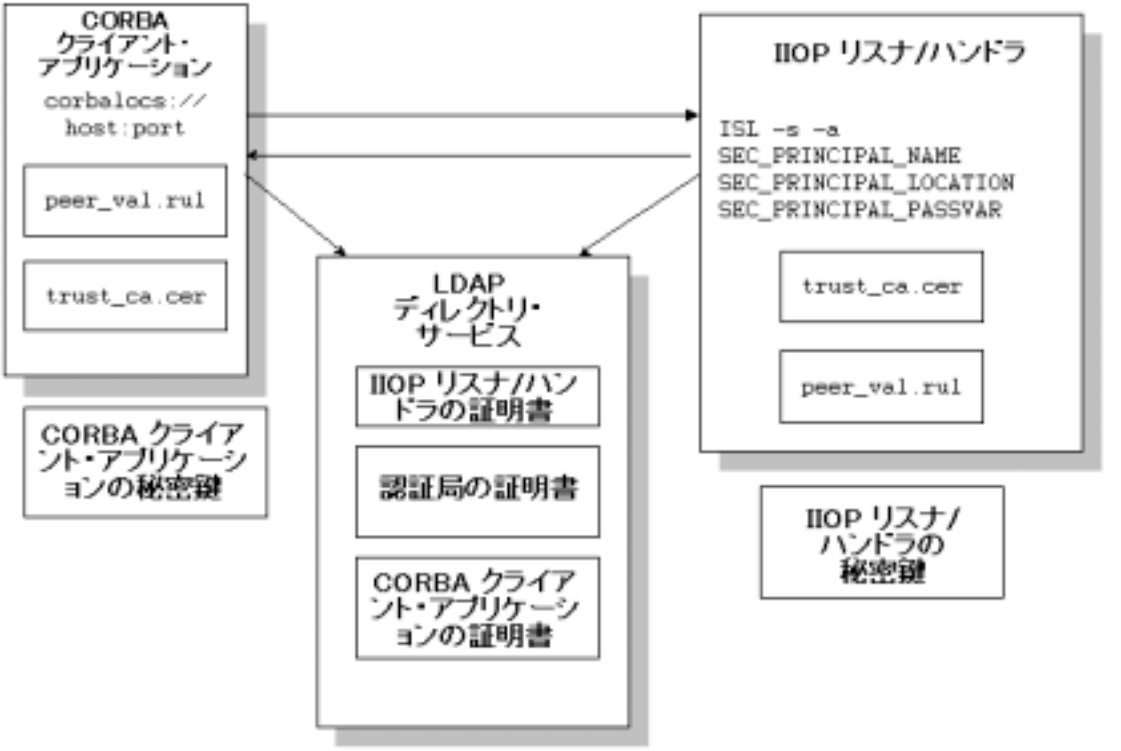


表 3-6 は、CORBA アプリケーションで証明書による認証を使用するためのプログラミング手順をまとめたものです。詳細については、第 10 章の 1 ページ「セキュリティをインプリメントする CORBA アプリケーション」を参照してください。

表 3-6 証明書による認証のプログラミング手順

手順	説明
1	Bootstrap オブジェクトの URL アドレス形式 (`corbaloc` または `corbalocs`) を使用するアプリケーション・コードを記述します。IIOP リスナ / ハンドラの証明書の識別名の CommonName が URL アドレス形式で指定されたホスト名と正確に一致している必要があります。URL アドレス形式の詳細については、第 10 章の -1 ページ「ブートストラップ処理メカニズムの使用」を参照してください。 CORBA INS ブートストラップ処理メカニズムを使用しても、BEA Tuxedo ドメインの PrincipalAuthenticator オブジェクトへのリファレンスを取得できます。CORBA INS の詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』を参照してください。
2	`SecurityLevel2::PrincipalAuthenticator` インターフェイスの `authenticate()` メソッドを使用するアプリケーション・コードを記述します。メソッドの引数として `Tobj::CertificateBased` を指定し、`Security::Opaque` の `auth_data` 引数として秘密鍵のパス・フレーズを指定します。

認証プラグインの使い方

BEA Tuxedo 製品では、認証プラグインを CORBA アプリケーションに統合することができます。BEA Tuxedo 製品は、共有シークレット・パスワード、ワンタイム・パスワード、CHALLENGE 応答、Kerberos などの認証技術を使用して、さまざまな認証プラグインに対応できます。認証インターフェイスは、必要に応じて GSSAPI (Generic Security Service Application Programming Interface) に基づいており、認証プラグインが GSSAPI に合わせて記述されていることを想定しています。

認証プラグインを使用する場合、BEA Tuxedo システムのレジストリで認証プラグインをコンフィギュレーションする必要があります。レジストリの詳細については、第 9 章の 1 ページ「セキュリティ・プラグインのコンフィギュレーション」を参照してください。

認証プラグインの詳細 (インストール手順およびコンフィギュレーション手順を含む) については、BEA 社の営業担当者にお問い合わせください。

認可

認可の機能により、管理者は CORBA アプリケーションへのアクセスを制御できます。つまり、管理者は、認可機能を使用して、CORBA アプリケーションのリソースまたはサービスに対するアクセス権をプリンシパル (認証されたユーザ) に許可するかどうかを決定します。

CORBA セキュリティ環境では、認可プラグインを統合できます。認可は、認証トークンで提示されたユーザ ID に基づいて決定されます。認証トークンは認証プロセスで生成されるので、認証プラグインと認可プラグインとの間で調整する必要があります。

認可プラグインを使用する場合、BEA Tuxedo システムのレジストリで認可プラグインをコンフィギュレーションする必要があります。レジストリの詳細については、第 9 章の 1 ページ「セキュリティ・プラグインのコンフィギュレーション」を参照してください。

認可プラグインの詳細 (インストール手順およびコンフィギュレーション手順を含む) については、BEA 社 の営業担当者にお問い合わせください。

監査

監査とは、操作要求とその結果に関する情報を収集、格納、および配布する方法です。監査証跡の記録からは、CORBA のセキュリティ方針に違反するアクションを実行したプリンシパルや、そのようなアクションを実行しようとしたプリンシパルを判別できます。また、これらの記録から、試行された操作、失敗した操作、および成功した操作を判別することもできます。

監査機能の現在のインプリメンテーションでは、ログオンの失敗、偽装の失敗、および許可されなかった操作を ULOG ファイルに記録できます。操作が許可されなかった場合、どの操作のパラメータの順序もデータ型もわからないので、操作のパラメータ値は提供されません。ログオンおよび偽装に関する監査のエントリには、認証を受けようとしたプリンシパルの ID が含まれます。ULOG ファイルの設定については、『BEA Tuxedo アプリケーションの設定』を参照してください。

監査プラグインを使用すると、CORBA アプリケーションの監査機能を拡張できます。BEA Tuxedo システムは、定義済みの実行ポイントで監査プラグインを呼び出します。通常、実行ポイントは、操作を試行する前、セキュリティ違反につながる現象を検出した場合、または操作が成功した場合です。監査情報を収集、処理、保護、および配布するためのアクションは、監査プラグインの機能によって異なります。監査情報を収集する場合はパフォーマンスへの影響を十分に考慮する必要があります。特に、成功した操作の監査には、大きなコストが発生することがあります。

監査に関する一部の決定は、監査トークンに格納されているユーザ ID に基づいて行われます。監査トークンは、認証プラグインによって生成されるため、認証プラグインと監査プラグインのプロバイダは、これらのプラグインが協調動作することを保証する必要があります。

監査要求の目的は、イベントを記録することです。各監査プラグインは、success (監査が成功し、イベントが記録される) または failure (監査は失敗し、イベントは記録されない) のどちらかの応答を返します。監査プラグインは、操作の実行前と実行後にそれぞれ一度呼び出されます。

- 操作前の監査では、操作を呼び出す前にも監査し、操作後のチェックのために入力データを保存することもできます。
- 操作後の監査では、操作の終了ステータスが報告されます。failure ステータスの場合、操作後の監査が呼び出されて、セキュリティ違反につながる現象を報告します。通常、こうしたレポートは、操作前または操作後の認可のチェックが失敗した場合や、セキュリティに対する攻撃が検出された場合に発行されます。

CORBA アプリケーションで監査プラグインの複数のインプリメンテーションを使用することができます。複数の認可プラグインを使用すると、操作前および操作後の監査操作が複数実行されます。

複数の監査プラグインを使用する場合、すべてのプラグインは、1 つのマスタ監査プラグインの下に配置されます。それぞれの下位認可プラグインは、SUCCESS または FAILURE を返します。いずれかのプラグインが操作に失敗すると、マスタ・プラグインが出力を FAILURE と決定します。その他のエラーも FAILURE と見なされます。それ以外の場合、SUCCESS が出力されます。

さらに、BEA Tuxedo システムのプロセスは、セキュリティ違反につながる現象を検出すると、監査プラグインを呼び出す場合があります。たとえば、操作前または操作後の認可のチェックが失敗したり、セキュリティに対する攻撃が検出された場合などです。これに対し、監査プラグインは、操作後の監査を実行して、監査が成功したかどうかを示す応答を返します。

BEA Tuxedo 製品に組み込まれている監査機能を使用する場合と、監査プラグインを使用する場合では、監査のプロセスが多少異なります。デフォルトの監査機能では、操作前の監査をサポートしていません。したがって、デフォルトの監査機能が、操作前の監査を行う要求を受け取っても、要求は直ちに返され、何も実行されません。

デフォルトの監査プラグイン以外の監査プラグインを使用する場合、BEA Tuxedo システムのレジストリでその認証プラグインをコンフィギュレーションする必要があります。レジストリの詳細については、第 9 章の 1 ページ「セキュリティ・プラグインのコンフィギュレーション」を参照してください。

監査プラグインの詳細 (インストール手順およびコンフィギュレーション手順を含む) については、BEA 社の営業担当者にお問い合わせください。

シングル・サイン・オン

シングル・サイン・オンを使用すると、WebLogic Server セキュリティ・レルムで認証された WebLogic Server ユーザが BEA Tuxedo ドメイン内の CORBA オブジェクトに対する要求を安全に送信することができます。シングル・サイン・オンがサポートされるのは、WebLogic Enterprise Connectivity によって提供される接続プールを使用し、その接続プールが CORBA 環境と信頼性のある関係を確立している場合に限定されます。プールが信頼性のある関係を確立する方法は次のいずれかです。

- パスワードによる認証を使用します。この場合、WebLogic Server ユーザは認証されますが、WebLogic Server レルムと BEA Tuxedo ドメイン間の要求は保護されません。
- パスワードによる認証と SSL プロトコルを使用します。この場合、SSL プロトコルを使用して、要求の整合性および機密性が保護されます。
- SSL プロトコルと証明書による認証を使用します。最も安全性の高い方法ですが、WebLogic Server および CORBA アプリケーションが公開鍵によるセキュリティ機能を実装する必要があります。

第 8 章の -1 ページ「シングル・サイン・オンのコンフィギュレーション」では、シングル・サイン・オンの各オプションを実装する方法を説明しています。

PKI プラグイン

BEA Tuxedo 製品には、CORBA アプリケーションでデジタル証明書を使用するために必要な SSL プロトコルとインフラストラクチャを含む PKI 環境が用意されています。ただし、PKI インターフェイスを使用して、カスタム・メッセージ・ベースのデジタル署名およびメッセージ・ベースの暗号化機能を CORBA アプリケーションに提供する PKI プラグインを統合できます。表 3-7 では PKI インターフェイスについて説明します。

表 3-7 PKI インターフェイス

PKI インターフェイス	説明
公開鍵の初期化	公開鍵ソフトウェアが公開鍵および秘密鍵を開けるようにします。たとえば、ゲートウェイ・プロセスでは、メッセージを復号化してから転送するために、特定の秘密鍵へのアクセスが必要なこともあります。

表 3-7 PKI インターフェイス (続き)

PKI インターフェイス	説明
鍵管理	公開鍵ソフトウェアが公開鍵および秘密鍵を管理および使用できるようにします。なお、メッセージ・ダイジェストとセッション・キーは、このインターフェイスを使用して暗号化および復号化されますが、公開鍵暗号を使用するバルク・データの暗号化は行われません。バルク・データの暗号化は、対称鍵暗号を使用して行われます。
証明ルックアップ	公開鍵ソフトウェアが、所定のプリンシパルに対する X.509v3 デジタル証明書を取得できるようにします。デジタル証明書は、Lightweight Directory Access Protocol (LDAP) など、適切な証明書リポジトリを使用して格納することができます。
証明解析	公開鍵ソフトウェアが、簡単なプリンシパル名と X.509v3 デジタル証明書を関連付けることができるようにします。パーサは、デジタル証明書を解析して、デジタル証明書に関連付けるプリンシパル名を生成します。
証明書の検証	公開鍵ソフトウェアが特定のビジネス・ロジックに基づいて X.509v3 デジタル証明書を検証することができます。
証明資料のマッピング	公開鍵ソフトウェアは、鍵を開くために必要な証明資料にアクセスしたり、認可トークンおよび監査トークンを提供したりすることができます。

PKI インターフェイスは、次のアルゴリズムをサポートしています。

- 公開鍵アルゴリズム :RSA (Rivest、Shamir、Adelman の頭文字を取ったもの) および Digital Signature Algorithm (DSA)
- 対称鍵アルゴリズム :
  - DES-CBC (Data Encryption Standard for Cipher Block Chaining)

- 2 つの鍵による Triple-DES
- Rivest’s Cipher 4 (RC4)

■ メッセージ・ダイジェスト・アルゴリズム :

- Message Digest 5 (MD5)
- Secure Hash Algorithm 1 (SHA-1)

PKI プラグインを使用する場合、BEA Tuxedo システムのレジストリで PKI プラグインをコンフィギュレーションする必要があります。レジストリの詳細については、第 9 章の 1 ページ「セキュリティ・プラグインのコンフィギュレーション」を参照してください。

PKI プラグインの詳細 (インストール手順およびコンフィギュレーション手順を含む) については、BEA 社の営業担当者にお問い合わせください。

CORBA のセキュリティ機能に関してよくある質問

ここでは、CORBA のセキュリティ機能についてよく寄せられる質問をいくつか取り上げて回答します。

既存の CORBA アプリケーションのセキュリティ機能を変更する必要がありますか

いいえ、必要ありません。CORBA アプリケーションで以前のバージョンの WebLogic Enterprise のセキュリティ・インターフェイスを使用している場合、CORBA アプリケーションを変更する必要はありません。現在のセキュリティ・スキーマをそのまま使用しても、既存の CORBA アプリケーションは BEA Tuxedo 8.0 製品でビルドした CORBA アプリケーションと協調動作します。

たとえば、CORBA アプリケーションが、接続したすべてのクライアント・アプリケーションに一般的な情報を提供する一連のサーバで構成されている場合、セキュリティ・スキーマを強化する必要はまったくありません。CORBA アプリケーションが、スニッファを検出できるだけのセキュリティ機能を持つ内部ネットワークのクライアント・アプリケーションに情報を提供する一連のサーバ・アプリケーションで構成されている場合、新たなセキュリティ機能を実装する必要はありません。

既存の CORBA アプリケーションで SSL プロトコルを使用できますか

はい、できます。SSL プロトコルを既存の CORBA アプリケーションで使用すると、さらにセキュリティ機能を強化することができます。たとえば、株価を特定のクライアント・アプリケーションに提供する CORBA サーバ・アプリケーションでは、SSL プロトコルを使用すると、クライアント・アプリケーションが正しい CORBA サーバ・アプリケーションに接続しており、不正なデータを持つ偽の CORBA サーバ・アプリケーションにルーティングされていないことを保証できます。クライアント・アプリケーションを認証する証明資料としては、ユーザ名とパスワードで十分です。しかし、SSL プロトコルを使用することで、メッセージの要求 / 応答情報をさらに高いレベルのセキュリティ機能で保護できます。

SSL プロトコルを使用すると、CORBA アプリケーションに次の利点があります。

- 初期ブートストラップ処理を含む会話全体を保護できます。SSL プロトコルは、介在者の攻撃、リプレイ攻撃、改ざん、およびスニッフィングを防ぐことができます。
- SSL プロトコルでは、最低 56 ビットの暗号化がデフォルト設定されているので、デフォルト設定をそのまま使用するだけでも、署名と暗号化による保護機能が提供されます。
- IIOP リスナ / ハンドラのデジタル証明書を使用して、接続されている IIOP リスナ / ハンドラをクライアント側で検証できます。クライアント・アプリケーションでは、セキュリティ規則を追加して、IIOP リスナ / ハンドラによるクライアント・アプリケーションへのアクセスを制限で

きます。コールバック・オブジェクトを使用すると、この保護機能はリモート・サーバ・アプリケーションに接続する IIOP リスナ / ハンドラにも適用できます。

CORBA アプリケーションで SSL プロトコルを使用するには、デジタル証明書を使用するインフラストラクチャを設定し、SSL プロトコルを使用するように ISL サーバ・プロセスのコマンド行オプションを変更した上で、IIOP リスナ / ハンドラの安全な通信用のポートをコンフィギュレーションします。既存の CORBA アプリケーションがパスワードによる認証を使用する場合、SSL プロトコルでそのコードを使用できます。CORBA C++ クライアント・アプリケーションが、Bootstrap オブジェクトへの初期リファレンスを解決して認証を実行する場合に、`InvalidDomain` 例外をキャッチしない場合は、この例外を処理するコードを記述します。詳細については、第 3 章の 25 ページ「シングル・サイン・オン」を参照してください。

注記 `Tobj_Bootstrap::resolve_initial_references()` メソッドの Java インプリメンテーションは、`InvalidDomain` 例外をスローしません。`corbaloc` または `corbalocs` URL アドレス形式を使用している場合、`Tobj_Bootstrap::resolve_initial_references()` メソッドは、内部で `InvalidDomain` 例外をキャッチし、`COMM_FAILURE` を例外としてスローします。このメソッドは、下位互換性を提供するためにこのように機能します。

証明書による認証をいつ使用すればいいですか

既存の CORBA アプリケーションを移行して、CORBA アプリケーションと市販の Web ブラウザ間でインターネット接続を使用するときです。たとえば、CORBA アプリケーションのユーザがインターネット上で買い物をするとします。ユーザに対して以下の点を保証する必要があります。

- アクセス先が目的のオンライン・ストアがあるサーバであって、クレジット・カード情報を盗むためにオンライン・ストアを装った偽のサーバではないこと。
- CORBA アプリケーションとオンライン・ストア間でやり取りするデータを、ネットワーク上の侵入者に知られないこと。

- オンライン・ストアとやり取りするデータが途中で変更されないこと。$500 相当の商品の注文が、悪意の有無に関わらず、$5000 の注文に変えられることがないこと。

こうした場合、SSL プロトコルおよび証明書による認証を使用すると、CORBA アプリケーションで最高レベルの保護機能が提供されます。SSL プロトコルによる利点以外にも、証明書による認証を使用すると、CORBA アプリケーションに以下の利点があります。

- クライアント・アプリケーションのデジタル証明書を使用して、IIOP リスナ / ハンドラ側でクライアント・アプリケーションを検証できます。また、IIOP リスナ / ハンドラでは、デジタル証明書の ID に基づいてクライアント・アプリケーションへのアクセスを制限する規則を追加できます。サーバ・アプリケーションとして機能するリモート ORB をコンフィギュレーションすると、相互認証を有効にして、デジタル証明書に基づいてクライアント・アプリケーションの ID を検証できます。

- BEA Tuxedo ドメイン内では、クライアント・アプリケーションに BEA Tuxedo ユーザ名およびパスワードを割り当てることができます。IIOP リスナ / ハンドラは、デジタル証明書に定義されている ID を BEA Tuxedo ユーザ名およびパスワードにマップするので、既存の CORBA アプリケーションはネイティブ CORBA サーバ・アプリケーション内に ID を持つことができます。

詳細については、第 3 章の 25 ページ「シングル・サイン・オン」を参照してください。

第 II 部セキュリティ機能の管理

章

4 公開鍵によるセキュリティ機能の管理

ここでは、以下の内容について説明します。

- 公開鍵によるセキュリティを使用するための要件
- デジタル証明書と公開鍵 / 秘密鍵のペアが必要な場合
- デジタル証明書の要求
- LDAP ディレクトリ・サービスでの証明書の公開
- LDAP 検索フィルタ・ファイルの編集
- 共通ロケーションにある秘密鍵
- 信頼性のある認証局の定義
- ピア規則ファイルの作成

ここで説明する作業は、CORBA アプリケーションで SSL プロトコルまたは証明書による認証を使用している場合にのみ実行してください。

公開鍵によるセキュリティを使用するための要件

SSL プロトコルおよび公開鍵によるセキュリティ機能を使用して、プリンシパルと BEA Tuxedo ドメイン間の通信を保護するには、専用のライセンスをインストールする必要があります。ライセンスのインストール方法については、『BEA Tuxedo システムのインストール』を参照してください。

また、公開鍵によるセキュリティを実装する前に、Lightweight Directory Access Protocol サーバと所属組織用の認証局 (市販または専用) を選択して設定する必要があります。

デジタル証明書と公開鍵 / 秘密鍵のペアが必要な場合

CORBA セキュリティ環境で SSL プロトコルを使用するには、秘密鍵と対応する公開鍵を含むデジタル署名付き証明書が必要です。必要なデジタル証明書および秘密鍵の数は、SSL プロトコルの使用方法によって異なります。

- リモート・クライアントと IIOP リスナ / ハンドラ間のネットワーク接続を保護するために SSL プロトコルを使用する場合、IIOP リスナ / ハンドラのデジタル証明書および秘密鍵を取得する必要があります。
- SSL プロトコルを証明書による認証で使用する場合、IIOP リスナ / ハンドラだけでなく、CORBA アプリケーションにアクセスする各プリンシパルのデジタル証明書および秘密鍵を取得する必要があります。

使用するデジタル証明書の取得先は、信頼性のある CA ファイルで定義されている認証局でなければなりません。詳細については、第 4 章の -8 ページ「信頼性のある認証局の定義」を参照してください。

デジタル証明書の要求

デジタル証明書を取得するには、デジタル署名要求 (CSR) と呼ばれる特定の形式でデジタル証明書の要求を提出する必要があります。CSR の作成方法は、利用する認証局によって異なります。通常、認証局では、公開鍵、秘密鍵、およびその公開鍵を含む CSR を提供します。CSR を作成するには、選択した認証局で説明されている手順に従います。

手順に従って CSR を作成すると、認証局から次のファイルを受け取ります。

ファイル	説明
key.der	秘密鍵ファイル。
request.pem	認証局に提出する CSR ファイル。内容は .dem ファイルと同じデータですが、電子メールにコピーしたり Web フォームに貼り付けたりできるように ASCII で符号化されています。

認証局からデジタル証明書を購入するには、認証局の登録手順に従って CSR を認証局に提出します。一部の認証局では、デジタル証明書を Web 経由で購入できます。

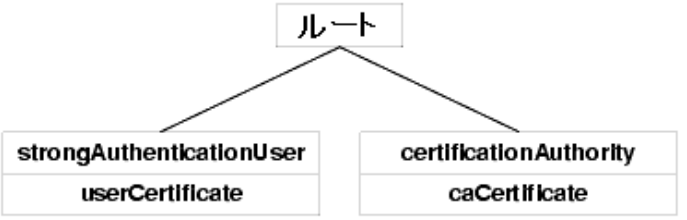
LDAP ディレクトリ・サービスでの証明書の公開

デジタル証明書の保存方法としては、グローバル・ディレクトリ・サービスを使用するのが一般的です。ディレクトリ・サービスを使用すると、増え続けるユーザに対してグローバルに利用可能にする情報を容易に管理できます。LDAP サーバでは、さまざまなディレクトリ・サービスにアクセスできます。

SSL プロトコルを使用するようにコンフィギュレーションされた BEA Tuxedo 製品の CORBA セキュリティ環境では、プリンシパルおよび認証局のデジタル証明書を、Netscape Directory Service や Microsoft Active Directory などの LDAP ディレクトリ・サービスから取り出すことができます。SSL プロトコルまたは認証局を使用する前に、LDAP ディレクトリ・サービスをインストールし、所属する組織に合わせてコンフィギュレーションする必要があります。BEA 社では、特定の LDAP ディレクトリ・サービスを提供しておらず、また推奨もしていません。ただし、選択した LDAP ディレクトリ・サービスは、X.500 スキーマ定義と LDAP バージョン 2 または 3 プロトコルをサポートしている必要があります。

LDAP ディレクトリ・サービスは、オブジェクト・クラスの階層を定義します。さまざまなオブジェクト・クラスがありますが、デジタル証明書に関連付けられるのは一部だけです。図 4-1 は、デジタル証明書に関連付けられる代表的なオブジェクト・クラスを示しています。

図 4-1 デジタル証明書の LDAP ディレクトリ構造



認証局から受け取ったデジタル証明書は、次のように LDAP ディレクトリ・サービスに保存します。

- IIOP リスナ / ハンドラおよびプリンシパルのデジタル証明書は、オブジェクト・クラスの `userCertificate` 属性を定義して、LDAP ディレクトリ・サービスに保存されます。通常、これらのデジタル証明書は、X.500 で定義されている `strongAuthenticationUser` オブジェクト・クラスのインスタンスとして保存されます。
- 認証局のデジタル証明書は、オブジェクト・クラスの `caCertificate` 属性を定義して、LDAP ディレクトリ・サービスに保存されます。通常、

これらのデジタル証明書は、X.500 で定義されている `certificateAuthority` クラスのインスタンスとして保存されます。

別のクラスを使用する LDAP スキーマでは、第 4 章の -5 ページ「LDAP 検索フィルタ・ファイルの編集」で説明するように、LDAP 検索ファイルを変更する必要があります。

BEA Tuxedo 製品では、デジタル証明書を Privacy Enhanced Mail (PEM) 形式でディレクトリ・サービスに保存する必要があります。

LDAP ディレクトリ・サービスを CORBA セキュリティ環境に統合する方法については、『BEA Tuxedo システムのインストール』を参照してください。

LDAP 検索フィルタ・ファイルの編集

SSL プロトコルまたは証明書による認証を使用するように CORBA アプリケーションをコンフィギュレーションする場合、LDAP 検索フィルタ・ファイルをカスタマイズして、ディレクトリ・サービスの検索スコープを限定するか、デジタル証明書を保持するオブジェクト・クラスを指定します。LDAP 検索フィルタ・ファイルをカスタマイズすると、性能が大幅に向上する場合があります。BEA Tuxedo 製品には、次の LDAP 検索フィルタが付属しています。

- 認証局に割り当てられているデジタル証明書用のディレクトリ・サービスを検索するフィルタ・スタンザ。このフィルタを使用すると、検索範囲が `certificationAuthority` オブジェクト・クラスのインスタンスに限定されます。
- プリンシパルに割り当てられているデジタル証明書用のディレクトリ・サービスを検索するフィルタ・スタンザ。このフィルタを使用すると、検索範囲が `strongAuthenticationUser` オブジェクト・クラスのインスタンスに限定されます。

ディレクトリ・サービスのスキーマがデジタル証明書を certificationAuthority および strongAuthenticationUser 以外のオブジェクト・クラスに保存するように定義されている場合、LDAP 検索フィルタ・ファイルを変更して、それらのオブジェクト・クラスを指定する必要があります。

LDAP 検索フィルタ・ファイルの場所は、BEA Tuxedo 製品のインストール時に指定します。詳細については、『BEA Tuxedo システムのインストール』を参照してください。

LDAP 検索フィルタ・ファイルは、管理者アカウントで所有する必要があります。このファイルを保護して、ファイルの所有者だけが読み書き特権を持つようにすることをお勧めします。

プリンシパルおよび認証局のデジタル証明書の検索を制限するには、次のタグで示される LDAP 検索フィルタ・ファイル内のフィルタ・スタンザを変更する必要があります。

- BEA_person_lookup
- BEA_issuer_lookup

これらのタグは、ディレクトリ・サービスで情報を検索するためのフィルタ式を含む LDAP 検索フィルタ・ファイル内のスタンザを識別します。これら BEA 固有のタグを使用すると、LDAP 検索フィルタ・ファイルのスタンザを、組織内のほかの LDAP 対応アプリケーションが使用する共通の LDAP 検索フィルタ・ファイルに保存できます。

BEA Tuxedo 製品が SSL プロトコルおよびデジタル証明書用に使用する LDAP 検索フィルタ・ファイルのスタンザの例を次に示します。

```
“BEA_person_lookup”
 “.*”  “ “ “(|(objectClass=strongAuthenticationUser) (mail=%v))”
                                        “e-mail address”
          “(|(objectClass=strongAuthenticationUser) (mail=%v))”
                                        “start of e-mail address”
“BEA_issuer_lookup”
 “.*” “ ” “(&(objectClass=certificationAuthority)
            (cn=%v)”  “exact match cn”
            (sn=%v))”   “exact match sn”
```

- BEA_person_lookup は、LDAP ディレクトリ・サービスで電子メール・アドレスを使用してプリンシパルを検索するように指定します。

- BEA_person_lookup には、LDAP ディレクトリ・サービスで一般名 (cn) を使用してプリンシパルを検索するように指定します。

LDAP 検索フィルタ・ファイルの詳細については、それぞれの LDAP 対応ディレクトリ・サービスのマニュアルを参照してください。

共通ロケーションにある秘密鍵

通常、プリンシパルが CSR を生成すると、秘密鍵の入ったファイルを受け取ります。プリンシパルが認証プロセスで 自身の ID を証明する場合には、この秘密鍵のファイルが必要になります。秘密鍵ファイルの所有者だけが読み取り特権を持ち、その他すべてのユーザはファイルにアクセスできないように、秘密鍵ファイルの保護機能を指定します。秘密鍵ファイルは、PEM 符号化された PKCS #8 保護形式で保存する必要があります。

BEA Tuxedo システムでは、以下のように、プリンシパルの電子メール・アドレスを使用して秘密鍵ファイルの名前を作成します。

1. 名前の @ は下線 (_) で置き換えられます。
2. ドット (.) 以降のすべての文字は削除されます。
3. .PEM ファイル拡張子がファイルに追加されます。

たとえば、プリンシパルの名前が milozzi@bigcompany.com の場合、生成される秘密鍵ファイルは milozzi_bigcompany.pem です。この命名規約では、ユーザ名が同じで電子メール・ドメインが異なる複数のプリンシパルが社内に存在してもかまいません。

BEA Tuxedo ソフトウェアでは、次のディレクトリで秘密鍵ファイルが検索されます。

Window 2000

```
%HOMEDRIVE%\%HOMEPATH%
```

UNIX

```
$HOME
```

また、BEA Tuxedo ソフトウェアでは、次のディレクトリでも秘密鍵ファイルが検索されます。

```
$TUXDIR/udataobj/security/keys
```

所有者だけが読み取り特権を持ち、その他すべてのユーザはディレクトリにファイルにアクセスできないように、`$TUXDIR\udataobj\security\keys` ディレクトリを保護する必要があります。

リスト 4-1 は、秘密鍵ファイルの一例です。

コード リスト 4-1 **秘密鍵ファイルの例**

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIICoDAaBgkqhkiG9w0BBQMwDQQItSFrtYcfKygCAQUEggKAEgrMxo8gYB/MOSXG
...
-----END ENCRYPTED PRIVATE KEY-----
```

信頼性のある認証局の定義

SSL 接続を確立する場合、CORBA のプロセス (クライアント・アプリケーション IIOP リスナ / ハンドラ) は、ピアのデジタル証明書のチェーンから認証局の ID および証明書を、信頼性のある認証局のリストと照合して、組織が信頼する認証局であることを確認します。このチェックは、Web ブラウザで行われるチェックとほぼ同じです。照合が失敗した場合、SSL 接続のイニシエータはターゲットの認証を拒否し、SSL 接続を終了します。通常は、システム管理者が、信頼性のある認証局のリストを定義します。

信頼性のある認証局のデジタル証明書を LDAP ディレクトリ・サービスから取り出します。PEM 形式のデジタル証明書を切り取って、`$TUXDIR\udataobj\security\certs` にある `trust_ca.cer` というファイルに貼り付けます。`trust_ca.cer` は、任意のテキスト・エディタで編集できます。

`trust_ca.cer` ファイルは、管理者アカウントで所有する必要があります。このファイルを保護して、ファイルの所有者だけが読み書き特権を持つようにすることをお勧めします。

リスト 4-2 は、信頼性のある認証局ファイルの一例です。

コード リスト 4-2 **信頼性のある認証局ファイルの例**

```
-----BEGIN CERTIFICATE----

MIIEuzCCBCSgAwIBAgIQKtZuM5AOzS9dZaIATJxIuDANBgkqhkiG9w0BAQQFADCB
zDEXMBUGA1UEChMOVmVyaVNpZ24sIEluYy4xHzAdBgNVBAsTFlZlcmlTaWduIFRy
dXN0IE5ldHdvcmsxRjBEBgNVBAsTPXd3dy52ZXJpc2lnbi5jb20vcmVwb3NpdG9y
eS9SUEEgSW5jb3JwLiBCeSBSZWYuLExJQUIuTFREKGMpOTgxSDBGBgNVBAMTP1Zl
cmlTaWduIENsYXNzIDEgQ0EgSW5kaXZpZHVhbCBTdWJzY3JpYmVyLVBlcnNvbmEg
...
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE----

MIIEuzCCBCSgAwIBAgIQKtZuM5AOzS9dZaIATJxIuDANBgkqhkiG9w0BAQQFADCB
zDEXMBUGA1UEChMOVmVyaVNpZ24sIEluYy4xHzAdBgNVBAsTFlZlcmlTaWduIFRy
dXN0IE5ldHdvcmsxRjBEBgNVBAsTPXd3dy52ZXJpc2lnbi5jb20vcmVwb3NpdG9y
...
-----END CERTIFICATE-----
```

ピア規則ファイルの作成

ネットワーク・リンク経由の通信では、接続先のピアが意図したピアまたは許可されたピアであることの確認が重要です。このチェックを行わないと、安全な接続を確立して、安全なメッセージをやり取りしたり、有効なデジタル証明書のチェーンを受信したりすることはできますが、介在者の攻撃を受ける危険は残ります。ピア検証を行うには、ピアの証明書に含まれている情報を、ピアの信頼性を検証するための規則を指定した情報リストと照合します。システム管理者は、このピア規則ファイルを管理します。

ピア規則は、`peer_val.rul` という ASCII ファイルで管理されます。`peer_val.rul` ファイルを、BEA Tuxedo ディレクトリ構造の次の場所に格納します。

```
$TUXDIR/udataobj/security/certs
```

リスト 4-3 は、ピア規則ファイルの一例です。

コード リスト 4-3 **ピア規則ファイルの例**

```
#
# このファイルには、ピアが安全な接続のターゲットとして認可されるかどうかを
# 検証するための規則のリストが含まれる
#
O=Ace Industry
O="Acme Systems, Inc."; OU=Central Engineering;L=Herkimer;S=NY
O="Ball, Corp.", C=US
o=Ace Industry, ou=QA, cn=www.ace.com
```

ピア規則ファイルの各規則は、キーで識別される要素のセットで構成されます。BEA Tuxedo 製品は、表 4-1 に示したキー名を認識します。

表 4-1 **ピア規則ファイルでサポートされているキー**

キー	属性
CN	CommonName
SN	SurName
L	LocalityName
S	StateOrProvinceName
O	OrganizationName
OU	OrganizationalUnitName
C	CountryName
E	EmailAddress

各キーの後ろには、オプションの空白文字、=、オプションの空白文字、および比較対象の値が続きます。キーでは大文字と小文字が区別されません。規則で指定された各要素がサブジェクトの識別名に入っており、要素の値が大文字と小文字の区別および区切りも含めて規則で指定された値と一致していない限り、規則は一致となりません。

ピア規則ファイルの各行には、安全な接続を確立するかどうかを決定するための 1 つの規則が入っています。規則は複数の行にまたがってはならず、規則全体を 1 行に収めなければなりません。規則の各要素を区切るには、カンマ (,) またはセミコロン (;) を使用します。

シャープ記号 (#) で始まる行は注釈です。注釈は、組織名と同じ行に表示できません。

次のいずれかの条件に該当する場合は、値を一重引用符で囲む必要があります。

- 文字列に次のいずれかの文字が含まれる場合

  , + = "" <CR> < > # ;

- 文字列の前または後ろに空白がある場合
- 文字列に空白が連続する場合

BEA Tuxedo 製品では、デフォルトで、ピア情報がピア規則ファイルと照合されます。このチェックを省略する場合は、空のピア規則ファイルを作成します。

章

5 リンク・レベルの暗号化のコンフィギュレーション

ここでは、次の内容について説明します。

- min 値と max 値について
- インストール済みの LLE バージョンの確認
- CORBA アプリケーション・リンクの LLE のコンフィギュレーション

min 値と max 値について

CORBA アプリケーションに LLE をコンフィギュレーションする前に、LLE の表記 (*min*, *max*) を知っておく必要があります。これらのパラメータのデフォルト値は次のとおりです。

- *min*: 0
- *max*: インストール済みの LLE バージョンで可能な暗号化の最高レベルを示すビット数

たとえば、米国およびカナダ版の LLE のデフォルトの *min* 値と *max* 値は (0, 128) です。デフォルト値を変更するには、アプリケーションの UBBCONFIG ファイルで *min* と *max* に新しい値を割り当てます。

インストール済みの LLE バージョンの確認

CORBA アプリケーションの *min* および *max* 値を設定する前に、マシンにインストールされている LLE のバージョンを確認する必要があります。マシンにインストールされた LLE バージョンを確認するには、次のように冗長モードで tmadmin コマンドを実行します。

```
tmadmin -v
```

BEA Tuxedo のライセンス・ファイル (lic.txt) のキー行は、リスト 5-1 のように記述されています。エントリ 128-bit Encryption Package は、米国およびカナダ版の LLE がインストールされていることを示します。

コード リスト 5-1 LLE **のライセンス情報**

```
INFO:  BEA Engine, Version 2.4
INFO:  Serial: 212889588, Expiration 2000-3-15, Maxusers 10000
INFO:  Licensed to: ACME CORPORATION
INFO:  128-bit Encryption Package
```

BEA Tuxedo のライセンス・ファイルは、次のディレクトリにあります。

Windows 2000

```
%TUXDIR%\udataobj\lic.txt
```

UNIX

```
$TUXDIR/udataobj/lic.txt
```

CORBA アプリケーション・リンクの LLE のコンフィギュレーション

CORBA アプリケーションで LLE をコンフィギュレーションするには、次のように、ネットワーク接続に参加する各 CORBA アプリケーションの UBBCONFIG ファイルで MINENCRYPTBITS および MAXENCRYPTBITS パラメータを定義する必要があります。

- MINENCRYPTBITS パラメータには、有効な最小ビット数を定義します。
- MAXENCRYPTBITS パラメータには、調整する暗号化の最大レベルを定義します。

MINENCRYPTBITS および MAXENCRYPTBITS パラメータに指定できる値は、0、40、および 128 です。ゼロを指定すると暗号化を使用しない設定となりますが、40 および 128 は暗号鍵で有効なビット数を指定します。

tmloadcf を実行してコンフィギュレーション・ファイルをロードします。tmloadcf コマンドを実行すると、UBBCONFIG が解析され、TUXCONFIG 変数が指す場所にバイナリ形式の TUXCONFIG ファイルがロードされます。

章

6 SSL プロトコルのコンフィギュレーション

ここでは、以下の内容について説明します。

- SSL プロトコルのパラメータの設定
- SSL ネットワーク接続用のポートの定義
- ホスト照合の有効化
- 暗号化レベルの設定
- セッション再調整の間隔の設定
- IIOP リスナ / ハンドラのセキュリティ・パラメータの定義
- ISL システム・プロセスのパラメータの設定例
- CORBA C++ ORB のコマンド行オプションの設定例

SSL プロトコルのパラメータの設定

IIOP リスナ / ハンドラまたは CORBA C++ オブジェクト・リクエスト・ブローカ (ORB) で SSL プロトコルを使用するには、以下の作業が必要です。

- SSL 接続を受け入れる安全なポートを指定します。
- データを暗号化する際のレベルを指定します。
- オプションで、セッションの再調整の間隔を設定します (IIOP リスナ / ハンドラのみ)。

以下の節では、ISL コマンドや CORBA C++ ORB のオプションを使用して SSL パラメータを設定する方法について説明します。

SSL ネットワーク接続用のポートの定義

SSL ネットワーク接続用のポートを定義するには、以下の手順に従います。

- ISL コマンドの `-s` オプションを使用して、IIOP リスナ / ハンドラが SSL プロトコルを使用した安全な接続をリッスンするポートを指定します。IIOP リスナ / ハンドラが SSL 接続だけを許可するようにコンフィギュレーションするには、ISL コマンドの `-s` オプションと `-n` オプションを同じ値に設定します。
- リモート CORBA C++ ORB を使用する場合は、ORB の `-ORBsecurePort` コマンド行オプションを使用して、ORB が SSL プロトコルを使用した安全な接続をリッスンするポートを指定します。コールバック・オブジェクトまたは CORBA ノーティフィケーション・サービスを使用する場合は、このコマンド行オプションを設定する必要があります。

注記　共同クライアント / サーバ・アプリケーションで SSL プロトコルを使用する場合は、SSL ネットワーク接続用のポートを指定しなければなりません。デフォルト設定を使用することはできません。

SSL ネットワーク接続用の安全なポートを定義するには、SSL プロトコルのライセンスをインストールしておく必要があります。`-S` オプションまたは `-ORBsecurePort` コマンド行オプションを実行したときに、SSL プロトコルを有効にするためのライセンスがインストールされていないと、IIOP リスナ / ハンドラまたは CORBA C++ ORB は起動しません。

ホスト照合の有効化

SSL プロトコルではメッセージを暗号化して機密性を保護できますが、暗号化は介在者の攻撃の防止策とはなりません。介在者の攻撃では、あるプリンシパルが、開始元アプリケーションがブートストラップ処理で使用する初期オブジェクト・リファレンスの取得先となる場所を偽ります。

介在者の攻撃を防ぐには、SSL 接続で受信したデジタル証明書が意図したプリンシパルのものであることを確認します。ホスト照合とは、SSL 接続を確立するために使用されたオブジェクト・リファレンスに指定されたホストが、ターゲットのデジタル証明書に指定された識別名のサブジェクトの一般名と一致するかどうかをチェックすることです。ホスト照合は、SSL 接続のイニシエータによってのみ実行され、要求のターゲットが実際にターゲットのデジタル証明書のドメイン名で指定されたネットワーク・アドレスにあることを確認されます。この照合が失敗した場合、SSL 接続のイニシエータはターゲットの認証を拒否し、SSL 接続を終了します。技術的には、ホスト照合は SSL の一部ではなく、Web ブラウザで行われるチェックに似ています。

デジタル証明書に含まれているドメイン名は、オブジェクト・リファレンスに含まれているホスト情報と正確に一致しなければなりません。したがって、IP アドレスの代わりに DNS ホスト名を使用することをお勧めします。

デフォルトでは、ホスト照合は、IIOP リスナ / ハンドラおよび CORBA C++ ORB で有効になっています。ホスト照合を有効にする必要がある場合は、次のいずれかの手順を実行します。

- IIOP リスナ / ハンドラでは、ISL コマンドの -v オプションを指定します。
- CORBA C++ ORB では、–ORBpeerValidate コマンド行オプションを指定します。

-v オプションおよび -ORBpeerValidate コマンド行オプションの値は、以下のとおりです。

- none ― ホスト照合は実行されません。
- detect ― SSL 接続を確立するためのオブジェクト・リファレンスがターゲットのデジタル証明書のホスト名と一致しない場合、IIOP リスナ / ハンドラまたは ORB はターゲットを認証できないので、SSL 接続を終了します。detect は、デフォルト値です。
- warn ― SSL 接続を確立するためのオブジェクト・リファレンスがターゲットのデジタル証明書のホスト名と一致しない場合、IIOP リスナ / ハンドラまたは ORB はメッセージをユーザ・ログに送信し、処理を続行します。

BEA Tuxedo ドメインで複数の IIOP リスナ / ハンドラが SSL 接続を使用するようにコンフィギュレーションされている場合 (障害耐久性のためなど)、IIOP リスナ / ハンドラの DNS エイリアス名を使用するか、それぞれの IIOP リスナ / ハンドラ用に別々のデジタル証明書を作成することをお勧めします。IIOP リスナの -H スイッチを使用すると、オブジェクト・リファレンスが正しく作成されるように、DNS エイリアス名を指定できます。

暗号化レベルの設定

暗号化レベルを設定するには、以下の手順に従います。

- ISL コマンドの -z および -Z オプションを使用して、IIOP リスナ / ハンドラでの暗号化のレベルを設定します。
- ORB の -ORBminCrypto および -ORBmaxCrypto コマンド行オプションを使用して、CORBA C++ ORB での暗号化のレベルを設定します。

-z オプションと -ORBminCrypto コマンド行オプションは、アプリケーションが IIOP リスナ / ハンドラまたは CORBA C++ ORB と SSL 接続を確立する場合に使用する暗号化の最低レベルを設定します。有効な値は、0、40、56、および 128 です。値 0 はデータが署名されるだけで暗号化されないことを示し、40、56、および 128 は暗号鍵の長さ (ビット単位) を指定します。暗号化の最低レベルを満たさない場合、SSL 接続は失敗します。デフォルト値は 40 です。

-Z オプションと -ORBmaxCrypto コマンド行オプションは、アプリケーションが IIOP リスナ / ハンドラまたは CORBA C++ ORB と SSL 接続を確立する場合に使用する暗号化の最大レベルを設定します。有効な値は、0、40、56、および 128 です。値 0 はデータが署名されるだけで暗号化されないことを示し、40、56、および 128 は暗号鍵の長さ (ビット単位) を指定します。デフォルトの最小値は 40 です。デフォルトの最大値は、ライセンスの指定に基づきます。

-z または -Z オプションと -ORBminCrypto および -ORBmaxCrypto コマンド行オプションは、SSL プロトコル用のライセンスがインストールされている場合にのみ有効です。

現在 CORBA アプリケーションで使用されている暗号化のレベルを変更するには、IIOP リスナ / ハンドラまたは ORB をシャットダウンする必要があります。

設定する暗号化の値の組み合わせは非常に重要です。SSL 接続のイニシエータに設定する暗号化の値は、SSL 接続のターゲットに設定する暗号化の値のサブセットである必要があります。

表 6-1 は、暗号化の値の組み合わせと暗号化の動作を示したものです。

表 6-1 暗号化の値の組み合わせ

-z -ORBminCrypto	-Z -ORBmaxCrypto	説明
指定なし	指定なし	ORBminCrypto および ORBmaxCrypto に値を指定せず、これら以外のコマンド行オプションまたはシステム・プロパティを使用して SSL プロトコルを使用するように指定した場合、こうしたコマンド行オプションやシステム・プロパティにはデフォルト値が設定されます。
0	指定なし	暗号化の最大レベルは、ライセンスで指定された最大値にデフォルト設定されます。改ざんやリプレイの検出および暗号化レベルが調整されます。
指定なし	0	改ざんやリプレイの検出が調整されます。暗号化は行われません。
0	0	改ざんやリプレイの検出が調整されます。暗号化は行われません。
40, 56, 128	指定なし	暗号化の最大レベルは、ライセンスで指定された最大値にデフォルト設定されます。暗号化レベルは、SSL ライセンスで許可されている最大レベルまで調整できます。
指定なし	40, 56, 12	暗号化レベルは、SSL ライセンスで許可される最大値より小さい値である限り、-Z オプションで指定された値まで調整できます。-z オプションのデフォルト値は 40 です。
40, 56, 128	40, 56, 128	暗号化レベルは、SSL ライセンスで許可される最大値より小さい値である限り、-z オプションで指定した値と -Z オプションで指定した値の間で調整できます。

注記　最大ビット・レベルは、表 6-1 に示した組み合わせの中で、SSL ライセンスの値によって制御されます。ラインセンスの最大値を超えてビット・レベルを指定した場合、IIOP リスナ / ハンドラまたは ORB は起動せず、ビット・レベルの設定が無効であることを示すメッセージが生成されます。最大値を引き下げて警告だけを表示する代わりに、IIOP リスナ / ハンドラまたは ORB を起動できないようにすることで、正しくコンフィギュレーションされていないアプリケーションを意図しているよりも低い保護レベルで実行せずに済みます。

ライセンスの最大ビット・レベルを超えて暗号化が調整された場合でも、SSL 接続は確立されません。

CORBA セキュリティ環境でサポートされている暗号スイートの一覧については、第 2 章の 12 ページ「サポートされている暗号スイート」を参照してください。

セッション再調整の間隔の設定

注記　セッションを再調整する間隔を設定するのは、IIOP リスナ / ハンドラの場合だけです。

ISL コマンドの -R オプションを使用して、セッションを再調整する間隔を制御します。SSL セッションを定期的に再調整すると、対称鍵の有効期間を制限する情報を暗号化 / 解読するための対称鍵が更新されます。暗号化用の対称鍵を定期的に変更することで、長時間の SSL 接続をより安全に継続できます。

-R オプションには、再調整の間隔を分単位で指定します。SSL 接続が指定した間隔で再調整を行う場合、IIOP リスナ / ハンドラは、インバウンド接続の SSL セッションの再調整をアプリケーションに要求するか、またはアウトバウンド接続の場合に実際に再調整を実行します。デフォルト値は 0 で、定期的なセッションの再調整は行われません。

ISL コマンドの -a オプションを使用して証明書による認証を有効にした場合は、セッションの再調整を使用できません。

IIOP リスナ / ハンドラのセキュリティ・パラメータの定義

SSL 接続に参加する場合、IIOP リスナ / ハンドラは SSL 接続を開始したピアに対して自身を認証します。この認証には、デジタル証明書が必要となります。デジタル証明書に関連付けられた秘密鍵は、SSL 接続を確立する処理の中で使用されます。この処理では、プリンシパルとピア (この場合は、クライアント・アプリケーションと IIOP リスナ / ハンドラ) がセッション・キーについて合意します。セッション・キーは、秘密鍵 / 公開鍵とは対照的に、SSL セッション中にデータを暗号化するための対称鍵です。ピアが認証できるように、IIOP リスナ / ハンドラに関して以下の情報を定義します。

- `SEC_PRINCIPAL_NAME`

  IIOP リスナ / ハンドラの ID を指定します。

- `SEC_PRINCIPAL_LOCATION`

  秘密鍵ファイルの場所を指定します。たとえば、`$TUXDIR\udataobj\security\keys\milozzi.pem` です。

- `SEC_PRINCIPAL_PASSVAR`

  `tmloadcf` コマンドを対話形式で実行しない場合に、IIOP リスナ / ハンドラの秘密鍵用のパス・フレーズを保持する環境変数を指定します。それ以外の場合は、`tmloadcf` コマンドを入力するときにパス・フレーズの入力を求められます。

注記　IIOP リスナ / ハンドラのセキュリティ・パラメータの定義が不正な場合、ULOG ファイルで次のエラーが報告されます。

```
ISH.28014: LIBPLUGIN_CAT:2008:ERROR:No such file or
directory SEC_PRINCIPAL_LOCATION
ISH.28014:ISNAT_CAT:1552:ERROR:Could not open private key,
erro =-3011
ISH.28104:ISNAT_CAT:1544:ERROR:Could not perform SSL accept
from host/port//IPADDRESS:PORT
```

このエラーを解決するには、セキュリティ・パラメータの情報を修正し、IIOP リスナ / ハンドラを再起動します。

これらのパラメータは、ISL システム・プロセスを定義する UBBCONFIG ファイルの SERVERS セクションに含まれています。

また、tpusradd コマンドを使用して、IIOP リスナ / ハンドラを BEA Tuxedo ドメインで認可されたユーザとして定義する必要があります。IIOP リスナ / ハンドラのパスワードの入力を求められます。SEC_PRINCIPAL_PASSVAR で定義したパス・フレーズを入力します。

初期化時に、IIOP リスナ / ハンドラは、SEC_PRINCIPAL_NAME で定義されたプリンシパル名を引数としてインクルードして、クリデンシャルを取得するための認証プラグインを呼び出します。CORBA アプリケーションと対話を試みるリモート・クライアント・アプリケーションを認証し、リモート・クライアント・アプリケーションの認証および監査トークンを取得できるように、IIOP リスナ / ハンドラにはクリデンシャルが必要です。

IIOP リスナ / ハンドラは、BEA Tuxedo ドメインに対して自身の ID を認証しないと、信頼性のあるシステム・プロセスになることができません。そのため、デフォルトの認証プラグインを使用する場合の認証サーバをコンフィギュレーションする必要があります。詳細については、第 7 章の -2 ページ「認証サーバのコンフィギュレーション」を参照してください。

ISL システム・プロセスのパラメータの設定例

ISL サーバ・プロセスに関する情報を定義する UBBCONFIG の SERVERS セクションで SSL プロトコルのパラメータを設定します。リスト 6-1 は、SSL プロトコルおよび証明書による認証を使用する IIOP リスナ / ハンドラのコンフィギュレーションのパラメータを設定する UBBCONFIG ファイルのコードです。

コード リスト 6-1 UBBCONFIG ファイルでの ISL コマンドの使い方

```
...
ISL
```

```
SRVGRP = SYS_GRP
SRVID  = 5
CLOPT  = “-A -- -a -z40 -Z128 -S3579 -n //ICEPICK:2569
SEC_PRINCIPAL_NAME=”BLOTTO”
SEC_PRINCIPAL_LOCATION=”BLOTTO.pem”
SEC_PRINCIPAL_VAR=”AUDIT_PASS”
```

CORBA C++ ORB のコマンド行オプションの設定例

リスト 6-2 のサンプル・コードは、SSL プロトコルを使用するように ORB をコンフィギュレーションするための CORBA C++ ORB のコマンド行オプションの使い方を示しています。

コード リスト 6-2 CORBA C++ ORB **のコマンド行オプションの設定例**

```
ChatClient    -ORBid BEA_IIOP
              -ORBsecurePort 2100
              -ORBminCrypto 40
              -ORBMaxCrypto 128
              TechTopics
```

章

7 認証のコンフィギュレーション

ここでは、以下の内容について説明します。

- 認証サーバのコンフィギュレーション
- 認可されたユーザの定義
- セキュリティ・レベルの定義
- アプリケーション・パスワードによるセキュリティのコンフィギュレーション
- パスワードによる認証のコンフィギュレーション
- パスワードによる認証用の UBBCONFIG ファイルの例
- 証明書による認証のコンフィギュレーション
- 証明書による認証用の UBBCONFIG ファイルの例
- アクセス制御のコンフィギュレーション
- 旧バージョンの WebLogic Enterprise クライアント・アプリケーションと相互運用するためのセキュリティのコンフィギュレーション

認証サーバのコンフィギュレーション

注記　認証サーバをコンフィギュレーションする必要があるのは、SECURITY パラメータに USER_AUTH 以上の値を指定し、デフォルトの認証プラグインを使用する場合だけです。

認証を行うには、ユーザの個々のパスワードを正当なユーザのファイルと照合することでユーザを認証するための認証サーバをコンフィギュレーションする必要があります。BEA Tuxedo システムでは、AUTHSRV というデフォルトの認証サーバを使用して認証を実行します。AUTHSVR は、認証を実行する AUTHSVC というサービスだけを提供します。セキュリティ・レベルが ACL または MANDATORY_ACL に設定されている場合、AUTHSVR サーバは、AUTHSVC を AUTHSVC として宣言します。

CORBA アプリケーションがユーザを認証するには、UBBCONFIG ファイルの RESOURCES セクションの AUTHSVC パラメータの値として、CORBA アプリケーションの認証サーバとして使用するプロセスの名前を指定する必要があります。サービスは、AUTHSVC と呼ばれます。AUTHSVC パラメータを UBBCONFIG ファイルの RESOURCES セクションに指定した場合、SECURITY パラメータも少なくとも USER_AUTH に指定する必要があります。この値を指定しない場合、tmloadcf コマンドが実行されたときにエラーが発生します。-m オプションを UBBCONFIG ファイルの ISL プロセスでコンフィギュレーションする場合、AUTHSVC は、ISL プロセスの前に、UBBCONFIG ファイルに定義する必要があります。

また、UBBCONFIG ファイルの SERVERS セクションで AUTHSVR を定義する必要があります。SERVERS セクションには、CORBA アプリケーションで起動するサーバ・プロセスに関する情報が格納されます。AUTHSVC をアプリケーションに追加するには、UBBCONFIG ファイルで、認証サービスとして AUTHSVC を定義し、認証サーバとして AUTHSVR を定義する必要があります。リスト 7-1 は、UBBCONFIG ファイルで認証サーバを定義する部分です。

コード リスト 7-1　認証サーバのパラメータ

```
*RESOURCES
SECURITY    USER_AUTH
```

```
AUTHSVC    “AUTHSVC”
   .
   .
   .

*SERVERS
AUTHSVR SRVGRP="group_name" SRVID=1 RESTART=Y GRACE=600 MAXGEN=2
CLOPT="-A"
```

パラメータと値の組み合わせである AUTHSVC のエントリを省略すると、BEA Tuxedo システムはデフォルトで AUTHSVC を呼び出します。

AUTHSVR は、アプリケーション固有のロジックをインプリメントする認証サーバと置き換えることができます。たとえば、広く使用されている Kerberos のメカニズムを使用して認証を行うため、認証サーバをカスタマイズすることもできます。

カスタマイズした認証サービスをアプリケーションに追加するには、UBBCONFIG ファイルで認証サービスと認証サーバを定義する必要があります。たとえば、次のように入力します。

```
*RESOURCES
SECURITY   USER_AUTH
AUTHSVC    KERBEROS
   .
   .
   .

*SERVERS
KERBEROSSVR SRVGRP="group_name" SRVID=1 RESTART=Y GRACE=600
MAXGEN=2 CLOPT="-A"
```

デフォルトの認証サーバをコンフィギュレーションしたら、IIOP リスナ / ハンドラの ID (UBBCONFIG ファイルの SEC_PRINCIPAL_NAME パラメータで指定) を tpusr ファイルで指定する必要があります。また、CORBA アプリケーションのすべてのユーザを tpusr ファイルで指定する必要があります。詳細については、第 7 章の -4 ページ「認可されたユーザの定義」を参照してください。

認可されたユーザの定義

CORBA アプリケーションのセキュリティ・コンフィギュレーションの中で、CORBA アプリケーションに対するアクセス権を持つプリンシパルおよびプリンシパル・グループを定義する必要があります。

認可されたユーザの定義方法は、次のとおりです。

- パスワードによる認証の場合、認可されたユーザを定義するにはパスワードと関連パスワードを使用します。
- 証明書による認証の場合、認可されたユーザを識別するには電子メール・アドレスを使用します。電子メール・アドレスによって、デジタル証明書で表されるプリンシパルの外部 ID が CORBA アプリケーションで使用される ID にマップされます。

認可されたプリンシパルのリストが格納されたファイルを作成するには、`tpusradd` コマンドを使用します。`tpusradd` コマンドは、新しいプリンシパルを BEA Tuxedo のセキュリティ・データ・ファイルに追加します。この情報は、認証サーバがプリンシパルを認証する場合に使用します。プリンシパルが格納されたファイルは `tpusr` と呼ばれます。

ファイルはコロン区切りのフラットな ASCII ファイルで、CORBA アプリケーションのシステム管理者だけが読み取り可能です。システム・ファイルのエントリは、1 行あたり 512 文字までです。ファイルはアプリケーション・ディレクトリに格納され、環境変数の `$APPDIR` で指定されます。環境変数 `$APPDIR` には、CORBA アプリケーションのパス名を設定する必要があります。

`tpusradd` ファイルは、管理者アカウントで所有する必要があります。このファイルを保護して、ファイルの所有者だけが読み書き特権を持つようにすることをお勧めします。

`tpusradd` コマンドのオプションは以下のとおりです。

- `-u` *`uid`*

  ユーザの ID 番号。UID は、128K 以下の正の 10 進整数でなければなりません。また、アプリケーションの既存の識別子リスト内で一意でなけ

ればなりません。UID は、次に利用可能な (一意な) 0 より大きい識別子にデフォルト設定されます。

- `-g gid`

  グループの ID 番号。GID は、整数識別子または文字列名です。このオプションは、新しいユーザのグループ・メンバーシップを定義します。デフォルトでは、other グループ (識別子 0) となります。

- `-c client_name`

  プリンシパル名を指定する出力可能な文字列。名前には、コロン (:)、シャープ記号 (#)、改行文字 (\n) を使用できません。プリンシパル名は CORBA アプリケーションの既存のプリンシパル・リスト内で一意でなければなりません。

- `usrname`

  ユーザの新規ログイン名を指定する出力可能な文字列。名前には、コロン (:)、シャープ記号 (#)、改行文字 (\n) を使用できません。ユーザ名は CORBA アプリケーションの既存のユーザ・リスト内で一意でなければなりません。

デフォルトの認証サーバを使用する場合、IIOP リスナ / ハンドラの ID (UBBCONFIG ファイルの SEC_PRINCIPAL_NAME パラメータで指定) を tpusr ファイルで指定する必要があります。また、CORBA アプリケーションのすべてのユーザを tpusr ファイルで指定する必要があります。

カスタム認証サービスを使用する場合、IIOP リスナ / ハンドラおよび CORBA アプリケーションのユーザを、カスタム認証サービスのユーザ・レジストリで定義します。また、tpusr というファイルが $APPDIR に指定されていてはなりません。この名前のファイルが指定されていた場合、CORBA/NO_PERMISSION 例外が生成されます。

リスト 7-2 に、tpusr ファイルの例を示します。

コード リスト 7-2 tpusr **ファイルの例**

```
Usrname   Cltname   Password Entry   Uid    GID

milozzi   “bar”              2        100    0
smart     “ “                1        1      0
```

```
pat       “tpsysadmin”        3        0      8192
butler     “tpsysadmin”        3         N/A     8192
```

注記　BEA Tuxedo のセキュリティ・データ・ファイルにプリンシパル・グループを追加するには、tpgrpadd コマンドを使用します。

tpusradd および tpgrpadd コマンド以外にも、BEA Tuxedo 製品には、tpusr および tpgrp ファイルを変更するための次のコマンドが用意されています。

- tpusrdel
- tpusrmod
- tpgrpdel
- tpgrpmod

各コマンドの詳細については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo コマンド・リファレンス』を参照してください。

ホスト・システムには、ユーザとグループのリストを格納したファイルが既に存在する場合があります。これらのファイルを CORBA アプリケーションのユーザ・ファイルおよびグループ・ファイルとして使用するには、BEA Tuxedo システムで受け付けられる形式に変換する必要があります。ファイルを変換するには、次の手順例に示すように、tpaclcvt コマンドを実行します。この手順例は、UNIX ホスト・マシン用に記述されています。

1. アプリケーションの MASTER マシンで作業しており、アプリケーションが非アクティブであることを確認します。
2. /etc/password ファイルを BEA Tuxedo システムで受け付けられる形式に変換するには、次のコマンドを入力します。

   ```
   tpaclcvt -u /etc/password
   ```

   このコマンドにより、tpusr ファイルが作成され、変換されたデータが格納されます。tpusr ファイルが既に存在する場合、tpaclcvt により、変換したデータがファイルに追加されます。ただし、重複するユーザ情報が追加されることはありません。

注記　シャドウ・パスワード・ファイルを使用するシステムでは、ファイル内のユーザごとにパスワードの入力が要求されます。

3. /etc/group ファイルを BEA Tuxedo システムで受け付けられる形式に変換するには、次のコマンドを入力します。

```
tpaclcvt -g /etc/group
```

このコマンドにより、tpgrp ファイルが作成され、変換されたデータが格納されます。tpgrp ファイルが既に存在する場合、tpaclcvt により、変換したデータがファイルに追加されます。ただし、重複するユーザ情報が追加されることはありません。

セキュリティ・レベルの定義

CORBA アプリケーションのセキュリティ定義の中で、UBBCONFIG ファイルの RESOURCES セクションの SECURITY パラメータを定義する必要があります。SECURITY パラメータの形式は次のとおりです。

```
*RESOURCES
       SECURITY {NONE|APP_PW|USER_AUTH|ACL|MANDATORY_ACL}
```

表 7-1 では、SECURITY パラメータの値について説明します。

表 7-1 SECURITY パラメータの値

値	説明
NONE	CORBA アプリケーションでパスワードまたはアクセスのチェックを行わないことを示します。 Tobj::PrincipalAuthenticator::get_auth_type() は、値 TOBJ_NOAUTH を返します。
APP_PW	BEA Tuxedo ドメインにアクセスするには、クライアント・アプリケーションはアプリケーション・パスワードを提供する必要があることを示します。tmloadcf コマンドでは、アプリケーション・パスワードの入力が要求されます。 Tobj::PrincipalAuthenticator::get_auth_type() は、値 TOBJ_SYSAUTH を返します。

表 7-1 SECURITY **パラメータの値** (続き)

値	説明
USER_AUTH	クライアント・アプリケーションと IIOP リスナ / ハンドラは BEA Tuxedo ドメインに対して自身を認証する必要があることを示します。値 USER_AUTH は APP_PW と似ていますが、クライアントの初期化時にユーザ認証も行われることを示します。tmloadcf コマンドでは、アプリケーション・パスワードの入力が要求されます。 Tobj::PrincipalAuthenticator::get_auth_type() は、値 TOBJ_APPAUTH を返します。 このセキュリティ・レベルでは、アクセス制御のチェックは行われません。
ACL	CORBA アプリケーションで認証が使用され、インターフェイス、サービス、キュー名、およびイベント名に対するアクセス制御のチェックが行われることを示します。名前に ACL が関連付けられていなければ、アクセス権が付与されていると見なされます。tmloadcf コマンドでは、アプリケーション・パスワードの入力が要求されます。 Tobj::PrincipalAuthenticator::get_auth_type returns a value of TOBJ_APPAUTH.
MANDATORY_ACL	CORBA アプリケーションで認証が使用され、インターフェイス、サービス、キュー名、およびイベント名に対するアクセス制御のチェックが行われることを示します。値 MANDATORY_ACL は ACL と似ていますが、名前に ACL が関連付けられていなければ、アクセスは拒否されます。tmloadcf コマンドでは、アプリケーション・パスワードの入力を要求されます。 Tobj::PrincipalAuthenticator::get_auth_type は、値 TOBJ_APPAUTH を返します。

注記　IIOP リスナ / ハンドラが証明書による認証を使用するようにコンフィギュレーションされている場合、SECURITY パラメータの値は USER_AUTH 以上でなければなりません。

アプリケーション・パスワードによるセキュリティのコンフィギュレーション

アプリケーション・パスワードによるセキュリティをコンフィギュレーションするには、以下の手順に従います。

1. アプリケーションの MASTER マシンで作業しており、アプリケーションが非アクティブであることを確認します。
2. UBBCONFIG ファイルの RESOURCES セクションの SECURITY パラメータに APP_PW をコンフィギュレーションします。
3. tmloadcf コマンドを実行してコンフィギュレーションをロードします。tmloadcf コマンドを実行すると、UBBCONFIG が解析され、TUXCONFIG 変数が指す場所にバイナリ形式の TUXCONFIG ファイルがロードされます。
4. パスワードの入力を求められます。パスワードは、30 文字以内で構成します。これはアプリケーションのパスワードとなり、tmadmin コマンドの passwd パラメータを使用して変更しない限り有効です。
5. 電話や手紙など、オンライン以外の方法で、認可されたユーザに対してアプリケーション・パスワードを配布します。

パスワードによる認証のコンフィギュレーション

パスワードによる認証では、CORBA アプリケーションと対話するため、各クライアントは、アプリケーション・パスワードのほか、有効なユーザ名とユーザ固有のデータ (パスワードなど) を提示しなければなりません。パスワードは、tpusr ファイルに保存されているユーザ名に関連付けられたパス

ワードと一致する必要があります。ユーザ・パスワードと、tpusr 内のユーザ名 / パスワードとの照合は、認証サーバ AUTHSVR の認証サービス AUTHSVC によって実行されます。

パスワードによる認証を有効にするには、以下の手順に従います。

1. ユーザとその関連パスワードを tpusr ファイルで定義します。tpusr ファイルの詳細については、第 7 章の 4 ページ「認可されたユーザの定義」を参照してください。
2. アプリケーションの MASTER マシンで作業しており、アプリケーションが非アクティブであることを確認します。
3. テキスト・エディタで UBBCONFIG を開き、RESOURCES セクションと SERVERS セクションに次の行を追加します。

```
*RESOURCES
SECURITY    USER_AUTH
AUTHSVC     “AUTHSVC”
      .
      .
      .

*SERVERS
AUTHSVR SRVGRP="group_name" SRVID=1 RESTART=Y GRACE=600 MAXGEN=2
CLOPT="-A"
```

   CLOPT="-A" を指定すると、tmboot コマンドは、tmboot によってアプリケーションが起動するときに、"-A" で呼び出されたデフォルトのコマンド行オプションだけを AUTHSVR に渡します。

4. tmloadcf コマンドを実行してコンフィギュレーションをロードします。tmloadcf コマンドを実行すると、UBBCONFIG が解析され、TUXCONFIG 変数が指す場所にバイナリ形式の TUXCONFIG ファイルがロードされます。
5. パスワードの入力を求められます。パスワードは、30 文字以内で構成します。これはアプリケーションのパスワードとなり、tmadmin コマンドの passwd パラメータを使用して変更しない限り有効です。
6. 電話や手紙など、オンライン以外の方法で、認可されたユーザに対してアプリケーション・パスワードを配布します。

パスワードによる認証用の UBBCONFIG ファイルの例

リスト 7-4 は、パスワードによる認証を使用するアプリケーション用の UBBCONFIG ファイルを示しています。UBBCONFIG ファイルの重要なセクションは、太字で表記されています。

コード リスト 7-3 **パスワードによる認証用の** UBBCONFIG **ファイルの例**

```
*RESOURCES
    IPCKEY    55432
    DOMAINID  securapp
    MASTER    SITE1
    MODEL     SHM
    LDBAL     N
    SECURITY  USER_AUTH
    AUTHSVR “AUTHSVC”

*MACHINES
    "ICEAXE"
    LMID        = SITE1
    APPDIR      = "D:\TUXDIR\samples\corba\SECURAPP"
    TUXCONFIG   = "D:\TUXDIR\samples\corba\SECURAPP\results
                     \tuxconfig"
    TUXDIR      = "D:\Tux8"
    MAXWSCLIENTS = 10

*GROUPS
    SYS_GRP
       LMID    = SITE1
       GRPNO   = 1
    APP_GRP
       LMID    = SITE1
       GRPNO   = 2

*SERVERS
    DEFAULT:
    RESTART = Y
    MAXGEN  = 5
```

```
AUTHSVR
    SRVGRP  = SYS_GRP
    SRVID   = 1
    RESTART = Y
    GRACE   = 60
    MAXGEN   = 2

TMSYSEVT
    SRVGRP  = SYS_GRP
    SRVID   = 1

TMFFNAME
    SRVGRP  = SYS_GRP
    SRVID   = 2
    CLOPT   = "-A -- -N -M"

TMFFNAME
    SRVGRP  = SYS_GRP
    SRVID   = 3
    CLOPT   = "-A -- -N"

TMFFNAME
    SRVGRP  = SYS_GRP
    SRVID   = 4
    CLOPT   = "-A -- -F"

simple_server
    SRVGRP  = APP_GRP
    SRVID   = 1
    RESTART = N

ISL
    SRVGRP  = SYS_GRP
    SRVID   = 5
    CLOPT    = “-A -- -n //PCWIZ::2500”
    SEC_PRINCIPAL_NAME="IIOPListener"
    SEC_PRINCIPAL_PASSVAR="ISH_PASS"
```

証明書による認証のコンフィギュレーション

証明書による認証では SSL プロトコルを使用するので、SSL プロトコル用のライセンスをインストールし、SSL プロトコルをコンフィギュレーションしてからでないと、証明書による認証を利用できません。SSL プロトコル用のライセンスのインストールについては、『BEA Tuxedo システムのインストール』を参照してください。SSL プロトコルのコンフィギュレーション方法については、第 6 章の 1 ページ「SSL プロトコルのコンフィギュレーション」を参照してください。

また、CORBA アプリケーションで証明書による認証を使用する前に、LDAP 対応のディレクトリおよび認証局を用意する必要があります。LDAP 対応であれば、どのディレクトリ・サービスでもかまいません。また、CORBA アプリケーションで使用する証明書および秘密鍵の取得先の認証局を選択する必要があります。詳細については、第 4 章の 1 ページ「公開鍵によるセキュリティ機能の管理」を参照してください。

証明書による認証を有効にするには、以下の手順に従います。

1. SSL プロトコルを使用するためのライセンスをインストールします。
2. LDAP 対応ディレクトリ・サービスをコンフィギュレーションします。
3. IIOP リスナ / ハンドラの証明書および秘密鍵を認証局から取得します。
4. CORBA アプリケーションの証明書および秘密鍵を認証局から取得します。
5. CORBA アプリケーションの秘密鍵をユーザの `Home` ディレクトリまたは次のディレクトリに格納します。

   Windows 2000

   ```
   %TUXDIR%\udataobj\security\keys
   ```

   UNIX

   ```
   $TUXDIR/udataobj/security/keys
   ```

6. IIOP リスナ / ハンドラ、CORBA アプリケーション、および認証局の証明書を LDAP 対応ディレクトリ・サービスで公開します。

7. ISL サーバ・プロセスの SEC_PRINCIPAL、SEC_PRINCIPAL_LOCATION、および SEC_PRINCIPAL_PASSVAR を UBBCONFIG ファイルで定義します。詳細については、第 6 章の 8 ページ「IIOP リスナ / ハンドラのセキュリティ・パラメータの定義」を参照してください。

8. tpusradd コマンドを使用して、CORBA アプリケーションおよび IIOP リスナ / ハンドラの認可されたユーザを定義します。tpusr ファイルでは、ユーザの電子メール・アドレスを使用します。tpusr ファイルの詳細については、第 7 章の 4 ページ「認可されたユーザの定義」を参照してください。IIOP リスナ / ハンドラのパスワードとして、SEC_PRINCIPAL_PASSVAR で定義したパス・フレーズを使用します。

9. ISL コマンドの -S オプションを使用して、安全な通信用の IIOP リスナ / ハンドラのポートを定義します。詳細については、第 6 章の 2 ページ「SSL ネットワーク接続用のポートの定義」を参照してください。

10. ISL コマンドの -a オプションを使用して、IIOP リスナ / ハンドラで証明書による認証を有効にします。

11. CORBA アプリケーションが信頼する認証局を定義する信頼性のある認証局ファイル (trust_ca.cer) を作成します。詳細については、第 4 章の 8 ページ「信頼性のある認証局の定義」を参照してください。

12. テキスト・エディタで UBBCONFIG を開き、RESOURCES セクションと SERVERS セクションに次の行を追加します。

```
*RESOURCES
SECURITY    USER_AUTH
```

13. tmloadcf コマンドを実行してコンフィギュレーションをロードします。tmloadcf コマンドを実行すると、UBBCONFIG が解析され、TUXCONFIG 変数が指す場所にバイナリ形式の TUXCONFIG ファイルがロードされます。

14. オプションで、CORBA アプリケーションおよび IIOP リスナ / ハンドラのピア規則ファイル (peer_val.rul) を作成します。詳細については、第 4 章の 9 ページ「ピア規則ファイルの作成」を参照してください。

15. オプションで、社内の階層に合わせて LDAP 検索フィルタ・ファイルを変更します。詳細については、第 4 章の 5 ページ「LDAP 検索フィルタ・ファイルの編集」を参照してください。

証明書による認証を有効にするには、次のいずれかを実行します。

- ISL コマンドの -a オプションを使用して、IIOP リスナ / ハンドラに接続するアプリケーションが証明書による認証を使用しなければならないことを指定します。
- ORB の -ORBmutualAuth コマンド行オプションを使用して、CORBA C++ ORB に接続するアプリケーションが証明書による認証を使用しなければならないことを指定します。

証明書による認証を有効にするには、SSL プロトコル用のライセンスをインストールしておく必要があります。-a オプションまたは -ORBmutualAuth コマンド行オプションを実行したときに、SSL プロトコルを有効にするためのライセンスがインストールされていないと、IIOP リスナ / ハンドラまたは CORBA C++ ORB は起動しません。

証明書による認証用の UBBCONFIG ファイルの例

リスト 7-4 は、証明書による認証を使用する CORBA アプリケーション用の UBBCONFIG ファイルを示しています。UBBCONFIG ファイルの重要なセクションは、太字で表記されています。

コード リスト 7-4 証明書による認証用の UBBCONFIG ファイルの例

```
*RESOURCES
    IPCKEY    55432
    DOMAINID  simpapp
    MASTER    SITE1
    MODEL     SHM
    LDBAL     N
```

```
    SECURITY  USER_AUTH
   AUTHSVR “AUTHSVC”


*MACHINES
    "ICEAXE"
    LMID        = SITE1
    APPDIR      = "D:\TUXDIR\samples\corba\SIMPAP~1"
    TUXCONFIG   = "D:\TUXDIR\samples\corba\SIMPAP~1
                  \results\tuxconfig"
    TUXDIR      = "D:\TUX8"
    MAXWSCLIENTS = 10

*GROUPS
    SYS_GRP
       LMID    = SITE1
       GRPNO   = 1
    APP_GRP
       LMID    = SITE1
       GRPNO   = 2

*SERVERS
    DEFAULT:
    RESTART = Y
    MAXGEN  = 5

    AUTHSVR
        SRVGRP  = SYS_GRP
        SRVID   = 1
        RESTART = Y
        GRACE   = 60
        MAXGEN   = 2

TMSYSEVT
        SRVGRP  = SYS_GRP
        SRVID   = 1

    TMFFNAME
        SRVGRP  = SYS_GRP
        SRVID   = 2
        CLOPT   = "-A -- -N -M"

    TMFFNAME
        SRVGRP  = SYS_GRP
        SRVID   = 3
        CLOPT   = "-A -- -N"

    TMFFNAME
        SRVGRP  = SYS_GRP
```

```
    SRVID   = 4
    CLOPT   = "-A -- -F"

simple_server
    SRVGRP  = APP_GRP
    SRVID   = 1
    RESTART = N

ISL
    SRVGRP  = SYS_GRP
    SRVID   = 5
    CLOPT   = "-A -- -a -z40 -Z128 -S2458 -n //ICEAXE:2468"
    SEC_PRINCIPAL_NAME="IIOPListener"
    SEC_PRINCIPAL_LOCATION="IIOPListener.pem"
    SEC_PRINCIPAL_PASSVAR="ISH_PASS"
```

アクセス制御のコンフィギュレーション

注記　アクセス制御の適用対象は、デフォルトの認可インプリメンテーションのみです。CORBA セキュリティ環境のデフォルトの認可プロバイダでは、アクセス制御のチェックが行われません。また、UBBCONFIG ファイルの SECURITY パラメータのコンフィギュレーションは、サード・パーティ製の認可インプリメンテーションで使用されるアクセス制御を管理または実施しません。

アクセス制御によるセキュリティには、オプションのアクセス制御リスト (ACL) と必須のアクセス制御リスト (MANDATORY_ACL) の 2 つのレベルがあります。アクセス制御リストは、ユーザがアプリケーションへの参加を認証された場合にのみ有効になります。

アクセス制御リストを使用すると、システム管理者はユーザを複数のグループにまとめ、それらのグループに対して、メンバ・ユーザがアクセス権を持つオブジェクトを関連付けることができます。アクセス制御は、次の理由により、グループ・レベルで行われます。

- システム管理を簡素化できます。新しいオブジェクトへのアクセス権を付与する場合、1 つのグループに対してアクセス権を付与する方が、個別の複数のユーザに付与するより簡単です。

- パフォーマンスを高めることができます。エンティティを呼び出すたびにアクセス・パーミッションをチェックする必要があるため、パーミッションはすばやく解決できなければなりません。グループの数はユーザの数より少ないため、特権ユーザのリストを検索するよりも、特権グループのリストを検索する方が高速です。

デフォルトの認可プロバイダを使用する場合、アクセス制御のチェック機能は、システム管理者が作成して管理する次のファイルに基づきます。

- tpusr にはユーザのリストが格納されています。
- tpgrp にはグループのリストが格納されています。
- tpacl には ACL のリストが格納されています。

オプションの ACL セキュリティのコンフィギュレーション

ACL と MANDATORY_ACL との違いは次のとおりです。

- ACL モードでは、サービス要求は特定の ACL がない場合に許可されます。
- MANDATORY_ACL モードでは、サービス要求は特定の ACL がない場合に拒否されます。

オプションの ACL セキュリティでは、各クライアントは、アプリケーションに参加するため、アプリケーション・パスワード、ユーザ名、およびユーザ固有のデータ (パスワードなど) を提示しなければなりません。

オプションの ACL セキュリティをコンフィギュレーションするには、次の手順に従います。

1. アプリケーションの MASTER マシンで作業しており、アプリケーションが非アクティブであることを確認します。
2. テキスト・エディタで UBBCONFIG を開き、RESOURCES セクションと SERVERS セクションに次の行を追加します。

```
*RESOURCES
SECURITY    ACL
AUTHSVC     “AUTHSVC”
      .
      .
      .
*SERVERS
AUTHSVR SRVGRP="group_name" SRVID=1 RESTART=Y GRACE=600 MAXGEN=2
CLOPT="-A"
```

CLOPT="-A" を指定すると、tmboot コマンドは、tmboot によってアプリケーションが起動するときに、"-A" で呼び出されたデフォルトのコマンド行オプションだけを AUTHSVR に渡します。デフォルトでは、AUTHSVR は tpusr ファイルのユーザ情報を使用して、CORBA アプリケーションと対話するクライアントを認証します。

3. tmloadcf コマンドを実行してコンフィギュレーションをロードします。tmloadcf コマンドを実行すると、UBBCONFIG が解析され、TUXCONFIG 変数が指す場所にバイナリ形式の TUXCONFIG ファイルがロードされます。
4. パスワードの入力を求められます。パスワードは、30 文字以内で構成します。これはアプリケーションのパスワードとなり、tmadmin の passwd コマンドを使用して変更しない限り有効です。
5. 電話や手紙など、オンライン以外の方法で、認可されたユーザに対してアプリケーション・パスワードを配布します。

必須の ACL セキュリティのコンフィギュレーション

必須の ACL セキュリティ・レベルでは、各クライアントは、CORBA アプリケーションと対話するため、アプリケーション・パスワード、ユーザ名、およびユーザ固有のデータ (パスワードなど) を提示しなければなりません。

必須の ACL セキュリティをコンフィギュレーションするには、以下の手順に従います。

1. アプリケーションの MASTER マシンで作業しており、アプリケーションが非アクティブであることを確認します。

2. テキスト・エディタで UBBCONFIG を開き、RESOURCES セクションと SERVERS セクションに次の行を追加します。

```
*RESOURCES
SECURITY   MANDATORY_ACL
AUTHSVC    ..AUTHSVC
      .
      .
      .
*SERVERS
AUTHSVR SRVGRP="group_name" SRVID=1 RESTART=Y GRACE=600 MAXGEN=2
CLOPT="-A"
```

   CLOPT="-A" を指定すると、tmboot コマンドは、tmboot によってアプリケーションが起動するときに、"-A" で呼び出されたデフォルトのコマンド行オプションだけを AUTHSVR に渡します。デフォルトでは、AUTHSVR は tpusr ファイルのクライアント・ユーザ情報を使用して、アプリケーションに参加するクライアントを認証します。tpusr ファイルは、アプリケーションの APPDIR 変数で定義されている最初のパス名が指すディレクトリにあります。

3. tmloadcf コマンドを実行してコンフィギュレーションをロードします。tmloadcf コマンドを実行すると、UBBCONFIG が解析され、TUXCONFIG 変数が指す場所にバイナリ形式の TUXCONFIG ファイルがロードされます。

4. パスワードの入力を求められます。パスワードは、30 文字以内で構成します。これはアプリケーションのパスワードとなり、tmadmin の passwd コマンドを使用して変更しない限り有効です。

5. 電話や手紙など、オンライン以外の方法で、認可されたユーザに対してアプリケーション・パスワードを配布します。

CORBA アプリケーション間の ACL 方針の設定

管理者は、次のコンフィギュレーション・パラメータを使用して、別々の BEA Tuxedo ドメインにある CORBA アプリケーション間のアクセス制御リスト (ACL) 方針をコンフィギュレーションおよび管理します。

パラメータ名	説明	設定
DMCONFIG の ACL_POLICY (DM_MIB の TA_DMACLPOLICY)	リモート・ドメイン・アクセス・ポイントごとに、DMCONFIG ファイルの DM_REMOTE_DOMAINS セクションで指定される場合があります。特定のリモート・ドメイン・アクセス・ポイントに対するこのパラメータの値により、ローカル・ドメイン・ゲートウェイがリモート・ドメインから受信したサービス要求の ID を変更するかどうかが決まります。*	LOCAL または GLOBAL。デフォルト値は LOCAL です。 LOCAL は、サービス要求の ID を変更することを示します。GLOBAL は、サービス要求を変更しないで渡すことを示します。DOMAINID 文字列は、リモート・ドメイン・アクセス・ポイントです。

* リモート・ドメイン・アクセス・ポイントは、RDOM (アールドム) または単に「リモート・ドメイン」とも呼ばれます。

以下では、ACL_POLICY のコンフィギュレーションが、ローカル・ドメイン・ゲートウェイ (GWTDOMAIN) のプロセスの動作に与える影響について説明します。

- ローカルの ACL 方針を使用する場合、各ドメイン・ゲートウェイ (GWTDOMAIN) がインバウンドの CORBA クライアント要求 (リモート・アプリケーションからネットワーク経由で受信される要求) を変更しています。変更された要求は、リモート・ドメイン・アクセス・ポイントに設定された DOMAINID の ID を持つため、その ID に設定されたアクセス権も取得することになります。各ドメイン・ゲートウェイは、アウトバウンドのクライアント要求を変更しないで渡します。

  このコンフィギュレーションでは、各アプリケーションに ACL データベースがあります。このデータベースには、ドメイン内のユーザに関するエントリだけが格納されます。

- グローバルの ACL 方針を使用する場合、各ドメイン・ゲートウェイ (GWTDOMAIN) は、インバウンドとアウトバウンドの CORBA クライアント要求を変更しないで渡します。このコンフィギュレーションでは、各

アプリケーションに ACL データベースがあります。このデータベースには、ドメイン内のユーザに関するエントリのほか、リモート・ドメインのユーザの情報も格納されます。

リモート・ドメイン・ゲートウェイの偽装化

ドメイン・ゲートウェイは、ローカルの DMCONFIG ファイルの ACL_POLICY パラメータに LOCAL(デフォルト) が設定されたリモート・ドメインからクライアント要求を受け取ると、要求からトークンを削除し、リモート・ドメイン・アクセス・ポイントの DOMAINID を含むアプリケーション・キーを作成します。

ACL 方針を指定する DMCONFIG のエントリ例

リスト 7-5 では、リモート・ドメイン・アクセス・ポイント b01 を介した接続に対し、ローカルの DMCONFIG ファイルで ACL がグローバルにコンフィギュレーションされています。つまり、ドメイン・アクセス・ポイント c01 のドメイン・ゲートウェイ・プロセスは、ドメイン・アクセス・ポイント b01 に対し、クライアント要求を変更しないで受け渡します。

コード リスト 7-5 ACL **方針を指定する** DMCONFIG **ファイルの例**

```
*DM_LOCAL_DOMAINS
# <LDOM name> <Gateway Group name> <domain type> <domain id>
#       [<connection principal name>] [<security>]...
c01     GWGRP=bankg1
        TYPE=TDOMAIN
        DOMAINID="BA.CENTRAL01"
        CONN_PRINCIPAL_NAME="BA.CENTRAL01"
        SECURITY=DM_PW
    .
    .
    .

*DM_REMOTE_DOMAINS
# <RDOM name> <domain type> <domain id> [<ACL policy>]
#       [<connection principal name>] [<local principal name>]...
b01     TYPE=TDOMAIN
        DOMAINID="BA.BANK01"
        ACL_POLICY=GLOBAL
        CONN_PRINCIPAL_NAME="BA.BANK01"
```

旧バージョンの WebLogic Enterprise クライアント・アプリケーションと相互運用するためのセキュリティのコンフィギュレーション

BEA Tuxedo ドメインの CORBA サーバ・アプリケーションを、WebLogic Enterprise 製品のリリース 4.2 および 5.0 で利用可能なセキュリティ機能を備えたクライアント・アプリケーションと安全に相互運用させることが必要になる場合があります。CORBA サーバ・アプリケーションを旧バージョンの安全なクライアント・アプリケーションと相互運用させるには、UBBCONFIG ファイルで CLOPT -t オプションを設定するか、CORBA オブジェクト・リクエスト・ブローカ (ORB) の -ORBinterOp コマンド行オプションを指定します。

CLOPT -t オプションを設定するか、-ORBinterOP コマンド行オプションを指定すると、CORBA サーバの有効なセキュリティ・レベルを引き下げることになります。したがって、互換モードをサーバ・アプリケーションで有効にする前に、十分に考慮する必要があります。

以前のクライアント・アプリケーションと相互運用するすべてのサーバ・アプリケーションの CLOPT -t オプションを設定する必要があります。CLOPT -t オプションは、UBBCONFIG ファイルの *SERVERS セクションで指定します。

コード リスト 7-6 **相互運用性を指定する** UBBCONFIG **ファイルのエントリ例**

```
*SERVERS
SecureSrv      SRVGRP=group_name SRVID=server_number
               CLOPT=A -t..
```

リモート CORBA C++ ORB を使用する場合、ORB の -ORBinterOp コマンド行オプションを指定して、ORB がリリース 4.2 または 5.0 の WebLogic Enterprise 製品のセキュリティ機能を使用するクライアント・アプリケーションと相互運用できるようにします。

章

8 シングル・サイン・オンのコンフィギュレーション

ここでは、以下の内容について説明します。

- パスワードによる認証とシングル・サイン・オン
- パスワードによる認証および SSL プロトコルとシングル・サイン・オン
- SSL プロトコルおよび証明書による認証とシングル・サイン・オン

パスワードによる認証とシングル・サイン・オン

パスワードによる認証でシングル・サイン・オンを実装する手順は、以下のとおりです。

1. `weblogic.properties` ファイルの `CORBA.connectionpool` セクションで、次のプロパティを定義します。
   - `appaddrlist=//`*`host`*`:`*`port`*

     `host` および *`port`* には、CORBA アプリケーションで使用する BEA Tuxedo ドメイン内の IIOP リスナ / ハンドラの名前およびポート番号

を指定します。CORBA アプリケーションでサポートされている各アドレス形式については、第 10 章の 1 ページ「セキュリティをインプリメントする CORBA アプリケーション」を参照してください。

- username には、WebLogic Server ユーザの名前を指定します。
- userpassword には、WebLogic Server ユーザのパスワードを指定します。
- apppassword には、アクセス先の CORBA アプリケーションのパスワードを指定します。
- securitycontext には、Yes を指定します。Yes は、WebLogic Server ユーザのセキュリティ・コンテキストを BEA Tuxedo ドメインに渡すことを示します。

注記　この他にも、weblogic.properties ファイルの CORBA.connectionpool セクションに接続プールを設定するためのプロパティがあります。CORBA 接続プールの設定方法については、WebLogic Server オンライン・マニュアルの『WebLogic Enterprise Connectivity ユーザーズ ガイド』を参照してください。

2. tpusradd コマンドを使用して、WebLogic Server ユーザを BEA Tuxedo ドメインで認可されたユーザとして定義します。WebLogic Server ユーザのユーザ名およびパスワードは、weblogic.properties ファイルで定義されているものを正確に tpusr ファイルに指定する必要があります。
3. IIOP リスナ / ハンドラが WebLogic Server セキュリティ・レルムから伝達されたセキュリティ・コンテキストを検出して利用するように、ISL コマンドの -E オプションをコンフィギュレーションします。ISL コマンドの -E オプションには、プリンシパル名を指定する必要があります。プリンシパル名は、weblogic.properties ファイルで定義されているユーザ名です。IIOP リスナ / ハンドラの ISL コマンドは、BEA Tuxedo ドメイン用の UBBCONFIG ファイルの CLOPT パラメータで定義されます。
4. UBBCONFIG ファイルの SECURITY パラメータを USER_AUTH 以上に設定します。

パスワードによる認証および SSL プロトコルとシングル・サイン・オン

パスワードによる認証および SSL プロトコルでシングル・サイン・オンを実装する手順は、以下のとおりです。

1. WebLogic Server および BEA Tuxedo CORBA 環境で SSL プロトコルをコンフィギュレーションします。

   WebLogic Server 環境での SSL プロトコルのコンフィギュレーション方法については、WebLogic Server オンライン・マニュアルの「セキュリティの管理」を参照してください。

   CORBA 環境での SSL プロトコルのコンフィギュレーション方法については、第 3 章の 25 ページ「シングル・サイン・オン」を参照してください。

2. `weblogic.properties` ファイルの `CORBA.connectionpool` セクションで、次のプロパティを定義します。
   - `appaddrlist=corbalocs://`*`host`*`:`*`port`*

     `host` および *`port`* には、アクセス先の BEA Tuxedo ドメイン内の IIOP リスナ / ハンドラの名前およびポート番号を指定します。CORBA アプリケーションでサポートされている各アドレス形式については、第 10 章の 1 ページ「ブートストラップ処理メカニズムの使用」を参照してください。
   - `username` には、WebLogic Server ユーザの名前を指定します。
   - `userpassword` には、WebLogic Server ユーザのパスワードを指定します。
   - `apppassword` には、アクセス先の CORBA アプリケーションのパスワードを指定します。
   - `securitycontext` には、`Yes` を指定します。Yes は、WebLogic Server ユーザのセキュリティ・コンテキストを BEA Tuxedo ドメインに渡すことを示します。

- minencryptionlevel および maxecryptionlevel。これらのプロパティは省略可能です。有効な値は 0、40、56、および 128 です。minencryptionlevel プロパティのデフォルト値は 40 です。maxecryptionlevel プロパティは、ライセンスで許可されている最大レベルにデフォルト設定されます。SSL ハンドシェーク時には、WebLogic Server 環境と BEA Tuxedo CORBA 環境間で使用する暗号化レベルがこの 2 つのプロパティに基づいて決定されます。

注記　このほかにも、weblogic.properties ファイルの CORBA.connectionpool セクションに CORBA 接続プールを設定するためのプロパティがあります。接続プールの設定方法については、WebLogic Server オンライン・マニュアルの『WebLogic Enterprise Connectivity ユーザーズ ガイド』を参照してください。

3. tpusradd コマンドを使用して、WebLogic Server ユーザを BEA Tuxedo ドメインで認可されたユーザとして定義します。WebLogic Server ユーザのユーザ名およびパスワードは、weblogic.properties ファイルで定義されているものを正確に tpusr ファイルに指定する必要があります。

4. IIOP リスナ / ハンドラが WebLogic Server セキュリティ・レルムから伝達されたセキュリティ・コンテキストを検出して利用するように、ISL コマンドの -E オプションをコンフィギュレーションします。ISL コマンドの -E オプションには、プリンシパル名を指定する必要があります。プリンシパル名は、weblogic.properties ファイルで定義されているユーザ名です。IIOP リスナ / ハンドラの ISL コマンドは、BEA Tuxedo ドメイン用の UBBCONFIG ファイルの CLOPT パラメータで定義されます。

5. UBBCONFIG ファイルの SECURITY パラメータを USER_AUTH 以上に設定します。

SSL プロトコルおよび証明書による認証とシングル・サイン・オン

SSL プロトコルおよび証明書による認証でシングル・サイン・オンを実装する手順は、以下のとおりです。

1. WebLogic Server および BEA Tuxedo CORBA 環境で SSL プロトコルをコンフィギュレーションします。

   WebLogic Server 環境での SSL プロトコルのコンフィギュレーション方法については、WebLogic Server オンライン・マニュアルの「セキュリティの管理」を参照してください。

   BEA Tuxedo CORBA 環境での SSL プロトコルのコンフィギュレーション方法については、第 3 章の 25 ページ「シングル・サイン・オン」を参照してください。

2. `weblogic.properties` ファイルの `CORBA.connectionpool` セクションで、次のプロパティを定義します。
   - `appaddrlist=corbalocs://`*`host`*`:`*`port`*

     `host` および *`port`* には、アクセス先の BEA Tuxedo ドメイン内の IIOP リスナ / ハンドラの名前およびポート番号を指定します。
   - `username` には、デジタル証明書のサブジェクトの電子メール・アドレスを指定します。
   - `userpassword` には、デジタル証明書の秘密鍵を指定します。
   - `apppassword` には、アクセス先の CORBA アプリケーションのパスワードを指定します。
   - `securitycontext` には、`Yes` を指定します。Yes は、WebLogic Server ユーザのセキュリティ・コンテキストを BEA Tuxedo ドメインに渡すことを示します。
   - `minencryptionlevel` および `maxecrptionlevel`。これらのプロパティは省略可能です。有効な値は 0、40、56、および 128 です。`minencryptionlevel` プロパティのデフォルト値は 40 です。

maxecryptionlevel プロパティは、ライセンスで許可されている最大レベルにデフォルト設定されます。SSL ハンドシェーク時には、WebLogic Server 環境と BEA Tuxedo CORBA 環境間で使用する暗号化レベルがこの 2 つのプロパティに基づいて決定されます。

- certificatebasedauth には、Yes を指定します。Yes は、証明書による認証を使用することを示します。

注記　このほかにも、weblogic.properties ファイルの CORBA.connectionpool セクションに CORBA 接続プールを設定するためのプロパティがあります。接続プールの設定方法については、WebLogic Server オンライン・マニュアルの『WebLogic Enterprise Connectivity ユーザーズ ガイド』を参照してください。

3. tpusradd コマンドを使用して、WebLogic Server ユーザを BEA Tuxedo ドメインで認可されたユーザとして定義します。WebLogic Server ユーザのユーザ名およびパスワードは、weblogic.properties ファイルで定義されているものを正確に tpusr ファイルに指定する必要があります。

4. IIOP リスナ / ハンドラが WebLogic Server セキュリティ・レルムから伝達されたセキュリティ・コンテキストを検出して利用するように、ISL コマンドの -E オプションをコンフィギュレーションします。ISL コマンドの -E オプションには、プリンシパル名を指定する必要があります。プリンシパル名は、weblogic.properties ファイルで定義されているユーザ名です。IIOP リスナ / ハンドラの ISL コマンドは、BEA Tuxedo ドメイン用の UBBCONFIG ファイルの CLOPT パラメータで定義されます。

5. IIOP リスナ / ハンドラが証明書による認証を有効にするように、ISL コマンドの -a オプションを設定します。IIOP リスナ / ハンドラの ISL コマンドは、BEA Tuxedo 用の UBBCONFIG ファイルの CLOPT パラメータで定義されます。

6. UBBCONFIG ファイルの SECURITY パラメータを USER_AUTH 以上に設定します。

WebLogic Server 環境と BEA Tuxedo CORBA 環境間で証明書による認証を使用するということは、WebLogic Server 環境から BEA Tuxedo CORBA 環境の CORBA オブジェクトへの接続を確立するために新しい SSL ハンド

シェークを実行するということです。同じ SSL ネットワーク接続を介して複数のクライアント要求をサポートするには、証明書による認証を以下のように設定する必要があります。

- WebLogic Enterprise Connectivity プロセス用のデジタル証明書を取得します。このデジタル証明書は、WebLogic Enterprise Connectivity プロセスの ID を認証するために、BEA Tuxedo CORBA 環境に提示されます。いったん接続が確立されると、WebLogic Enterprise Connectivity 製品と BEA Tuxedo 環境間の認証された接続は維持されます。
- WebLogic Server 環境から BEA Tuxedo CORBA 環境の CORBA に対するクライアント要求が送信されると、デジタル証明書が 2 つの環境間で交換され、接続の両端用のセッション・キーが生成されます。WebLogic Connectivity は WebLogic Server の一部なので、WebLogic Connectivity プロセスは、2 つの環境間で SSL 接続が確立されたときに作成されたセッション・キーを持つ BEA Tuxedo CORBA 環境からのメッセージをすべて受け入れます。WebLogic Enterprise Connectivity プロセスは、確立された SSL 接続を使用してクライアント要求を BEA Tuxedo 環境に転送します。

章

9 セキュリティ・プラグインのコンフィギュレーション

ここでは、セキュリティ・プラグイン (SPI) の登録について説明します。

セキュリティ・プラグイン (SPI) の登録

BEA Tuxedo 製品の CORBA および ATMI 環境では、セキュリティなどのコア・サービスのセットで構成される、共通のトランザクション処理のインフラストラクチャ (TP インフラストラクチャ) が使用されています。CORBA アプリケーションでは、適切に定義されたインターフェイスを介して TP インフラストラクチャを使用できます。これらのインターフェイスを使用して独自のサービス・コード・モジュール (セキュリティ・プラグイン) をロードし、リンクすることにより、システム管理者は、TP インフラストラクチャのデフォルトの動作を変更できます。

セキュリティ・プラグインを使用するには、そのセキュリティ・プラグインを BEA Tuxedo システムに登録する必要があります。BEA Tuxedo システムのレジストリは、セキュリティ・プラグインに関連する情報を格納しておくディスク・ベースのリポジトリです。このレジストリには、デフォルトのセキュリティ・プラグインに関する情報が最初に格納されています。カスタム・セキュリティ・プラグインを BEA Tuxedo システムに追加すると、エン

トリがレジストリに追加されます。セキュリティ・プラグインのレジストリ・エントリは、プラグインに関する情報を格納するバイナリ・ファイルのセットです。レジストリは、BEA Tuxedo システムごとに用意されています。CORBA アプリケーションの各クライアント・アプリケーション、サーバ・アプリケーション、およびサーバ・マシンは、同じセットのセキュリティ・プラグインを使用しなければなりません。

レジストリは次のディレクトリにあります。

Windows 2000

```
$TUXDIR\udataobj
```

UNIX

```
$TUXDIR/udataobj
```

カスタム・セキュリティ・プラグインを使用する CORBA アプリケーションのシステム管理者は、そのプラグインを登録する必要があります。BEA Tuxedo システムのレジストリへのセキュリティ・プラグインの登録は、ローカル・マシンからのみ可能です。つまり、システム管理者は、リモートからホスト・マシンにログオンしている間はセキュリティ・プラグインを登録できません。

セキュリティ・プラグインの管理では、次のコマンドを使用できます。

- `epifreg` ― セキュリティ・プラグインの登録
- `epifunreg` ― セキュリティ・プラグインの登録解除
- `epifregedt` ― レジストリ情報の編集

これらのコマンドの使い方については、『Developing Security Services for ATMI and CORBA Environments』を参照してください。このマニュアルでは、セキュリティ SPI の仕様を記載しており、セキュリティ・プラグインを動的にロードしてリンクすることを可能にする BEA Tuxedo プラグイン・フレームワークについて説明しています。このマニュアルの入手方法については、BEA 社の営業担当者にお問い合わせください。

カスタム・セキュリティ・プラグインをインストールする場合、プラグインのセキュリティ・ベンダは、カスタム・セキュリティ・プラグインにアクセスできるように BEA Tuxedo システムのレジストリを設定するためのコマンドの使い方を提供する必要があります。

第 III 部セキュリティのプログラミング

章

10 セキュリティをインプリメントするCORBA アプリケーション

ここでは、以下の内容について説明します。

- ブートストラップ処理メカニズムの使用
- パスワード認証の使用
- 証明書による認証の使用
- インターオペラブル・ネーミング・サービス・メカニズムの使用
- invocations_options_required() メソッドの使用

ブートストラップ処理メカニズムの使用

注記　このメカニズムは、BEA CORBA クライアント・アプリケーションで使用します。

BEA Tuxedo CORBA 環境の Bootstrap オブジェクトは機能拡張されており、指定した IIOP リスナ / ハンドラに対するすべての接続を保護するように指定できます。Bootstrap オブジェクトは、IIOP リスナ / ハンドラの場所を指定する際に使用する corbaloc および corbalocs の URL (Uniform Resource Locator) アドレス形式をサポートしています。提供されるセキュリティのタイプは、IIOP リスナ / ハンドラの場所の指定に使用する URL の形式によって異なります。

ホストとポートのアドレス形式の場合、URL アドレス形式を使用して IIOP リスナ / ハンドラの場所を指定しますが、ブートストラップ処理プロセスの動作は異なります。corbaloc または corbalocs URL アドレス形式を使用する場合、IIOP リスナ / ハンドラへの初期接続は、以下のいずれかに分かれます。

- プリンシパルは、Tobj::PrincipalAuthenticator::logon メソッドまたは SecurityLevel2::PrincipalAuthenticator::authenticate メソッドのいずれかによるパスワード認証を使用します。
- プリンシパルは、SecurityCurrent 以外のオブジェクト ID 値を使用して Tobj_Bootstrap::resolve_initial_references メソッドを呼び出します。

corbalocs URL アドレス形式を使用すると、最低限、プリンシパルと IIOP リスナ / ハンドラの間の接続の整合性を保護するために SSL プロトコルが使用されます。

表 10-1 では、2 つの URL アドレス形式の違いを説明します。

表 10-1 corbaloc および corbalocs URL アドレス形式の違い

URL アドレス形式	機能
corbaloc	デフォルトでは、IIOP リスナ / ハンドラに対する呼び出しは保護されません。オプションで、IIOP リスナ / ハンドラに対して SSL プロトコルをコンフィギュレーションします。 プリンシパルは、SecurityLevel2::PrincipalAuthenticator インターフェイスの authenticate() メソッドおよび SecurityLeve12::Credentials インターフェイスの invocation_options_required() メソッドを使用し、証明書による認証を使用するように指定してブートストラップ処理をセキュリティ保護します。
corbalocs	IIOP リスナ / ハンドラに対する呼び出しが保護され、IIOP リスナ / ハンドラまたは CORBA C++ ORB は、SSL プロトコルの使用を有効にするようにコンフィギュレーションする必要があります。詳細については、第 6 章の 1 ページ「SSL プロトコルのコンフィギュレーション」を参照してください。

corbaloc および corbalocs URL アドレス形式は、TCP/IP とドメイン・ネーム・システム (DNS) の両方の環境で簡単に操作できる文字列化されたオブジェクト・リファレンスを提供します。corbaloc および corbalocs URL アドレス形式には、DNS スタイルのホスト名または IP アドレスおよびポートが含まれます。

URL アドレス形式は、インターオペラブル・ネーミング・サービスの一部として Object Management Group (OMG) によって適用されたオブジェクト URL の定義に基づいて、その定義を拡張します。BEA Tuxedo ソフトウェアは、WebLogic Enterprise 製品の旧リリースの機能をサポートすることに加えて、セキュリティ保護された HTTP の URL に従ってモデル化された安全な形式をサポートするために、OMG インターオペラブル・ネーミング・サービスで記述された URL 形式も拡張します。

リスト 10-1 には、新しい URL アドレス形式の例があります。

コード リスト 10-1　corbaloc **および** corbalocs URL **アドレス形式の例**

```
corbaloc://555xyz.com:1024,corbaloc://555backup.com:1022,
corbaloc://555last.com:1999
corbalocs://555xyz.com:1024,(corbalocs://555backup.com:1022|corba
locs://555last.com:1999)
corbaloc://555xyz.com:1111
corbalocs://24.128.122.32:1011, corbalocs://24.128.122.34
```

BEA Tuxedo 製品は、異なるスキームの複数の URL の列挙をサポートする構文を拡張し、OMG インターオペラブル・ネーミング・サービスで記述した URL 構文の機能を向上しています。リスト 10-2 に、複数の URL を指定する例を示します。

コード リスト 10-2　**複数の** URL **アドレス形式の指定の例**

```
corbalocs://555xyz.com:1024,corbaloc://555xyz.com:1111
corbalocs://ctxobj.com:3434,corbalocs://mthd.com:3434,corbaloc://force.com:1111
```

リスト 10-2 の例では、パーサは、URL `corbaloc://force.com:1111` に達すると、安全な接続を試行しなかったものとして内部状態をリセットし、保護されていない接続を試行します。これは、クライアント・アプリケーションが Credential オブジェクトで SSL パラメータを設定していない場合に起こります。

以下の節では、Bootstrap オブジェクトの異なるアドレス形式を使用した場合の動作を説明します。

ホストとポートのアドレス形式の使用

CORBA クライアント・アプリケーションが、Bootstrap オブジェクトのホストとポートのアドレス形式を使用する場合、Bootstrap オブジェクトのコンストラクタ・メソッドは、指定したホスト名およびポート番号を基にオブジェクト・リファレンスを作成します。IIOP リスナ / ハンドラに対する呼び出しは、SSL プロトコルによる保護なしに行われます。

クライアント・アプリケーションは、パスワード認証による認証も実行できます。ただし、ブートストラップ処理は非保護および非検証リンクで実行されるので、すべての接続は、以下のセキュリティ攻撃に対して脆弱です。

- 介在者の攻撃。接続が実行されたプリンシパルが望ましいプリンシパルかどうかを検証できないという理由によります。
- サービス拒否攻撃。オブジェクト・リファレンスが返されない、返されたオブジェクト・リファレンスが無効、またはセキュリティ・トークンが無効という理由によります。
- スニッファ攻撃。情報を自由に送信できるので、パケット・スニッファを持つユーザが、暗号化されていない (たとえば、ユーザ名 / パスワード情報のみが暗号化されている) メッセージの内容を見ることができるという理由によります。
- 改ざん攻撃。情報の整合性が保護されないという理由によります。メッセージの内容が変更されても、その変更は検出されません。
- リプレイ攻撃。同じ要求が繰り返し送信されても、それが検出されないという理由によります。

注記　IIOP リスナ / ハンドラが SSL プロトコル用にコンフィギュレーションされ、Bootstrap オブジェクトのホストとポートのアドレス形式が使用されている場合、指定した CORBA オブジェクトに対する呼び出しによって `INVALID_DOMAIN` 例外が発生します。

corbaloc URL アドレス形式の使用

デフォルトでは、`corbaloc` URL アドレス形式およびパスワード認証を使用する場合に、IIOP リスナ / ハンドラに対する呼び出しは保護されていません。したがって、すべての通信は、以下のセキュリティ攻撃に対して脆弱です。

- 介在者の攻撃。接続が実行されたプリンシパルが望ましいプリンシパルかどうかを検証できないという理由によります。
- サービス拒否攻撃。オブジェクト・リファレンスが返されない、返されたオブジェクト・リファレンスが無効、またはセキュリティ・トークンが無効という理由によります。

- スニッファ攻撃。情報を自由に送信できるので、パケット・スニッファを持つユーザが、暗号化されていない (たとえば、ユーザ名 / パスワード情報のみが暗号化されている) メッセージの内容を見ることができるという理由によります。
- 改ざん攻撃。情報の整合性が保護されないという理由によります。メッセージの内容が変更されても、その変更は検出されません。
- リプレイ攻撃。同じ要求が繰り返し送信されても、それが検出されないという理由によります。

corbaloc URL アドレス形式を使用する際に、`SecurityLevel2::PrincipalAuthenticator::authenticate()` メソッドを使用して、証明書による認証を使用するように指定し、Credentials オブジェクトの `invocation_methods_required` メソッドを呼び出して、ブートストラップ処理を保護できます。

注記　IIOP リスナ / ハンドラが SSL プロトコル用にコンフィギュレーションされているものの、証明書による認証用にコンフィギュレーションされておらず、corbaloc URL アドレス形式が使用されている場合、指定した CORBA オブジェクトに対する呼び出しによって `INVALID_DOMAIN` 例外が発生します。

既存の CORBA アプリケーションを、ホストとポートのアドレス形式ではなく corbaloc URL 形式に移行することをお勧めします。

corbalocs URL アドレス形式の使用

corbalocs URL アドレス形式は、プリンシパルと IIOP リスナ / ハンドラの間の通信を保護する場合に推奨される形式です。corbalocs URL アドレス形式は、corbaloc URL アドレス形式とほぼ同じように機能しますが、使用する認証のタイプに関係なく、IIOP リスナ / ハンドラまたは CORBA C++ ORB を使用した通信を保護するために SSL プロトコルを使用する点が異なります。

デフォルトを corbalocs URL アドレス形式で使用する場合、通信は、サービス拒否攻撃に対してのみ脆弱です。SSL プロトコルおよび証明書による認証を使用すると、スニッファ、改ざん、リプレイ攻撃を防ぐことができます。さらに、デジタル証明書で指定したホストの検証によって、介在者の攻撃も防ぐことができます。

corbalocs URL アドレス形式を使用するには、SSL プロトコルの使用を有効にするように IIOP リスナ / ハンドラまたは CORBA C++ ORB をコンフィギュレーションする必要があります。IIOP リスナ / ハンドラまたは CORBA C++ ORB を SSL プロトコル用にコンフィギュレーションする際の詳細については、第 6 章の 1 ページ「SSL プロトコルのコンフィギュレーション」を参照してください。

パスワード認証の使用

ここでは、CORBA アプリケーションでのパスワード認証のインプリメントについて説明します。

Security サンプル・アプリケーション

Security サンプル・アプリケーションでは、パスワード認証を使用します。Security サンプル・アプリケーションでは、アプリケーションを使用している学生が固有の ID とパスワードを持っている必要があります。Security サンプル・アプリケーションは、以下のように動作します。

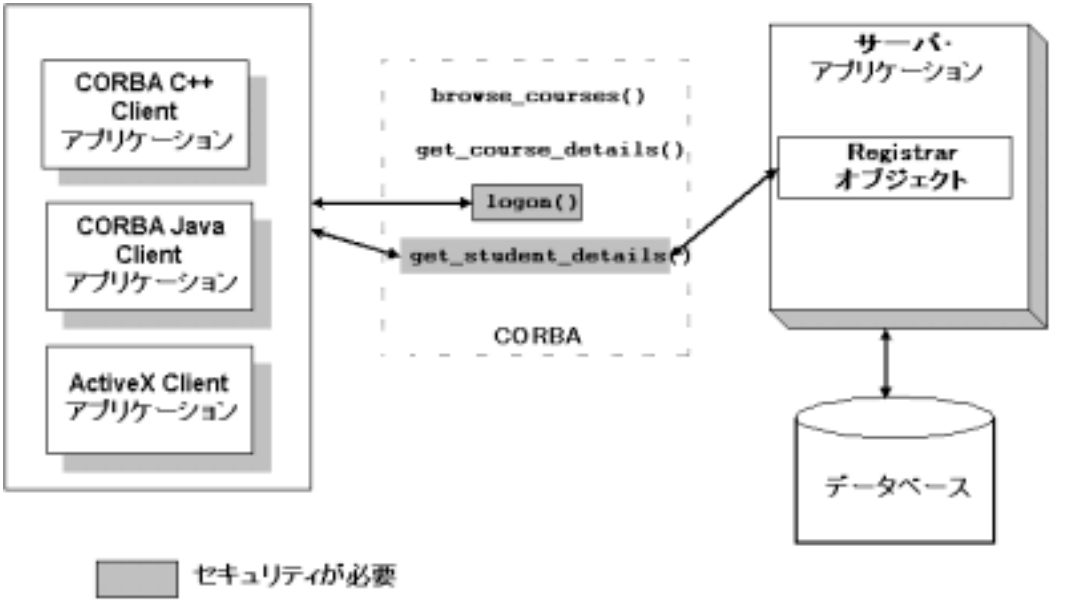
1. クライアント・アプリケーションには、ログオン・メソッドがあります。このオペレーションでは、ドメインにログオンする過程で取得される PrincipalAuthenticator オブジェクトのオペレーションが呼び出されます。
2. サーバ・アプリケーションは、Registrar オブジェクトの get_student_details() メソッドをインプリメントし、学生に関する情報を返します。ユーザが認証され、ログオンが完了すると、

`get_student_details()` メソッドは、データベース内の学生の情報にアクセスし、クライアントのログオン・メソッドが必要とする学生の情報を取得します。

3. Security サンプル・アプリケーションのデータベースには、コースと学生に関する情報が格納されています。

図 10-1 では、Security サンプル・アプリケーションのしくみを示します。

図 10-1Security サンプル・アプリケーション



Security サンプル・アプリケーションのソース・ファイルは、BEA Tuxedo ソフトウェアの `\samples\corba\university` ディレクトリにあります。Security サンプル・アプリケーションの作成と実行の詳細については、『BEA Tuxedo CORBA University サンプル・アプリケーション』を参照してください。

クライアント・アプリケーションの記述

パスワード認証を使用する場合、以下を実行するクライアント・アプリケーション・コードを記述します。

1. Bootstrap オブジェクトを使用し、指定した BEA Tuxedo ドメインで SecurityCurrent オブジェクトのリファレンスを取得します。ホストとポートのアドレス形式、`corbaloc` URL アドレス形式、または `corbalocs` URL アドレス形式を使用できます。

2. SecurityCurrent オブジェクトから PrincipalAuthenticator オブジェクトを取得します。
3. 以下のいずれかのメソッドを使用してプリンシパルを認証します。
   - C++ の場合 ― `Tobj::TuxedoSecurity` を使用した `SecurityLevel2::PrincipalAuthenticator::authenticate()`
   - Java の場合 ― `Tobj::TuxedoSecurity` を使用した `SecurityLevel2.PrincipalAuthenticator.authenticate()`
   - C++ の場合 ― `Tobj::PrincipalAuthenticator::logon()`
   - Java の場合 ― `Tobj.PrincipalAuthenticator.logon()`

`SecurityLevel2::PrincipalAuthenticator` インターフェイスは、CORBA サービス・セキュリティ・サービス仕様で定義されます。このインターフェイスには、プリンシパルの認証に使用する 2 つのメソッドがあります。メソッドが 2 つあるのは、プリンシパルの認証では複数の手順を必要とする可能性があるからです。`authenticate()` メソッドでは、呼び出し側は、認証を行い、必要に応じてこのセッションのプリンシパルの属性を選択できます。

CORBA 環境は、BEA Texudo 製品の ATMI 環境にあるものと同様のセキュリティをサポートする機能で PrincipalAuthenticator オブジェクトを拡張します。拡張機能は、`Tobj::PrincipalAuthenticator` インターフェイスによって提供されます。

`Tobj::PrincipalAuthenticator` インターフェイス用に定義されたメソッドは、同等の CORBA 定義のインターフェイスの特化および単純化した形式を提供します。CORBA アプリケーションを開発する場合、CORBA 定義の拡張か BEA Tuxedo 拡張のいずれかを使用できます。

`Tobj::PrincipalAuthenticator` インターフェイスは、`SecurityLevel2::PrincipalAuthenticator` インターフェイスと同じ機能を提供します。ただし、`SecurityLevel2::PrincipalAuthenticator::authenticate()` メソッドとは異なり、`Tobj::PrincipalAuthenticator` インターフェイスの `logon()` メソッドは、Credentials オブジェクトを返しません。結果として、複数のプリンシパル ID を使用する CORBA アプリケーションでは、ログオンの結果として Credentials オブジェクトを取得する `logon()` メソッドの直後に

Current::get_credentials() メソッドを呼び出す必要があります。ログオン・メソッドの直後の Credentials オブジェクトの取得は、シリアル化されたアクセスで保護する必要があります。

注記　ログオンの一部として指定したユーザ・データには、NULL を含めることはできません。

次の節には、パスワード認証のインプリメントを示す C++ および Java のコード例があります。Visual Basic のコード例については、第 17 章の 1 ページ「オートメーション・セキュリティ・リファレンス」を参照してください。

SecurityLevel2::PrincipalAuthenticator::authenticate() メソッドを使用する C++ コード例

リスト 10-3 には、SecurityLevel2::PrincipalAuthenticator::authenticate() メソッドを使用してパスワード認証を実行する C++ コード例があります。

コード リスト 10-3　SecurityLevel2::PrincipalAuthenticator::authenticate() **メソッドを使用する** C++ **クライアント・アプリケーション**

```
...
//Bootstrap オブジェクトを作成
     Tobj_Bootstrap* bootstrap = new Tobj_Bootstrap(orb,
                       corbalocs://sling.com:2143);

//SecurityCurrent オブジェクトを取得
CORBA::Object_var var_security_current_oref =
     bootstrap.resolve_initial_references(“SecurityCurrent”);
SecurityLevel2::Current_var var_security_current_ref =
     SecurityLevel2::Current::_narrow(var_security_current_oref.in());

//PrincipalAuthenticator を取得
SecurityLevel2::PrincipalAuthenticator_var var_principal_authenticator =
     var_security_current_oref->principal_authenticator();

const char * user_name = “john”
const char * client_name = “university”;
char system_password[31] = {‘\0’};
char user_password[31] = {‘\0’};
```

```
Tobj::PrincipalAuthenticator_ptr var_bea_principal_authenticator =
   Tobj::PrincipalAuthenticator::_narrow(var_bea_principal_authenticator.in());

// セキュリティ・レベルを決定
Tobj::AuthType auth_type = var_bea_principal_authenticator->get_auth_type();
switch (auth_type)
{
   case Tobj::TOBJ_NOAUTH;
   break;

   case Tobj::TOBJ_SYSAUTH
   strcpy(system_password, “sys_pw”);

   case Tobj::TOBJ_APPAUTH
    strcpy(system_password, “sys_pw”);
    strcpy(user_password, “john_pw”);
    break;
}
if (auth_type != Tobj::TOBJ_NOAUTH)

{
     SecurityLevel2::Credentials_var            creds;
     Security::Opaque_var                       auth_data;
     Security::AttributeList_var                privileges;
     Security::Opaque_var                       cont_data;
     Security::Opaque_var                       auth_spec_data;

var_bea_principalauthenticator->build_auth_data(user_name,
                                                client_name,
                                                system_password,
                                                user_password,
                                                NULL,
                                                auth_data,
                                                privileges);
Security::AuthenticationStatus status =
      var_bea_principalauthenticator->authenticate(
                                                   Tobj::TuxedoSecurity,
                                                   user_name,
                                                   auth_data,
                                                   privileges,
                                                   creds,
                                                   cont_data, auth_spec_data);

if (status != Security::SecAuthSuccess)
 {
    // 認証に失敗
    return;
```

```
 }
}

// アプリケーションに進む
...
```

Java Code Example That Uses the SecurityLevel2.PrincipalAuthenticator.authenticate() Method

リスト 10-4 には、SecurityLevel2.PrincipalAuthenticator.authenticate() メソッドを使用してパスワード認証を実行する Java コード例があります。

コード リスト 10-4 SecurityLevel2.PrincipalAuthenticator.authenticate() **メソッドを使用する** Java **クライアント・アプリケーション**

```
...
//Bootstrap オブジェクトを作成
      Tobj_Bootstrap bs =
         new Tobj_Bootstrap(orb, corbalocs://sling.com:2143);

//SecurityCurrent オブジェクトを取得
      org.omg.CORBA.Object secCurObj =
         bs.resolve_initial_references( "SecurityCurrent" );
      org.omg.SecurityLevel2.Current secCur2Obj =
         org.omg.SecurityLevel2.CurrentHelper.narrow(secCurObj);

      //Principal Authenticator を取得
      org.omg.Security.PrincipalAuthenticator princAuth =
         secCur2Obj.principal_authenticator();
      com.beasys.Tobj.PrincipalAuthenticator auth =
         Tobj.PrincipalAuthenticatorHelper.narrow(princAuth);

      // 認証タイプを取得
      com.beasys.Tobj.AuthType authType = auth.get_auth_type();

      // 引数を初期化
      String userName = "John";
      String clientName = "Teller";
      String systemPassword = null;
      String userPassword = null;
      byte[] userData = new byte[0];
```

```
            // 要求されたセキュリティ・レベルに従って引数を準備
            switch(authType.value())
              {
              case com.beasys.Tobj.AuthType._TPNOAUTH:
                 break;

              case com.beasys.Tobj.AuthType._TPSYSAUTH:
                systemPassword = "sys_pw";
                break;

              case com.beasys.Tobj.AuthType._TPAPPAUTH:
                systemPassword = "sys_pw";
                userPassword = "john_pw";
                break;
              }

            // セキュリティ・データを構築
            org.omg.Security.OpaqueHolder auth_data =
               new org.omg.Security.OpaqueHolder();
            org.omg.Security.AttributeListHolder privs =
               new Security.AttributeListHolder();
            auth.build_auth_data(userNname, clientName, systemPassword,
                                 userPassword, userData, authData,
                                 privs);

            // ユーザを認証
            org.omg.SecurityLevel2.CredentialsHolder creds =
               new org.omg.SecurityLevel2.CredentialHolder();
            org.omg.Security.OpaqueHolder cont_data =
               new org.omg.Security.OpaqueHolder();
            org.omg.Security.OpaqueHolder auth_spec_data =
               new org.omg.Security.OpaqueHolder();

            org.omg.Security.AuthenticationStatus status =
               auth.authenticate(com.beasys.Tobj.TuxedoSecurity.value,
                                 0, userName, auth_data.value(),
                                 privs.value(), creds, cont_data,
                                 auth_spec_data);
            if (status != AuthenticatoinStatus.SecAuthSuccess)
              System.exit(1);
            }
...
```

Tobj::PrincipalAuthenticator::logon() メソッドを使用する C++ コード例

リスト 10-5 には、Tobj::PrincipalAuthenticator::logon() メソッドを使用してパスワード認証を実行する C++ コード例があります。

コード リスト 10-5 Tobj::PrincipalAuthenticator::logon() **メソッドを使用する** C++ **クライアント・アプリケーション**

```
...
CORBA::Object_var var_security_current_oref =
     bootstrap.resolve_initial_references(“SecurityCurrent”);
SecurityLevel2::Current_var var_security_current_ref =
     SecurityLevel2::Current::_narrow(var_security_current_oref.in());

//PrincipalAuthenticator を取得
SecurityLevel2::PrincipalAuthenticator_var var_principal_authenticator_oref =
     var_security_current_oref->principal_authenticator();

//PrincipalAuthenticator をナロー変換
Tobj::PrincipalAuthenticator_var var_bea_principal_authenticator =
     Tobj::PrincipalAuthenticator::_narrow
                                     var_principal_authenticator_oref.in());


const char * user_name = “john”
const char * client_name = “university”;
char system_password[31] = {‘\0’};
char user_password[31] = {‘\0’};

// セキュリティ・レベルを決定
Tobj::AuthType auth_type = var_bea_principal_authenticator->get_auth_type();
switch (auth_type)
{
   case Tobj::TOBJ_NOAUTH;
   break;

   case Tobj::TOBJ_SYSAUTH
   strcpy(system_password, “sys_pw”);

   case Tobj::TOBJ_APPAUTH
    strcpy(system_password, “sys_pw”);
    strcpy(user_password, “john_pw”);
    break;
```

```
}
if (auth_type != Tobj::TOBJ_NOAUTH)

{
     SecurityLevel2::Credentials_var             creds;
     Security::Opaque_var                        auth_data;
     Security::AttributeList_var                 privileges;
     Security::Opaque_var                        cont_data;
     Security::Opaque_var                        auth_spec_data;

// セキュリティ・レベルを決定
Tobj::AuthType auth_type = var_bea_principal_authenticator->get_auth_type();
Security::AuthenticationStatus status = var_bea_principal_authenticator->logon(
                                user_name,
                                client_name,
                                system_password,
                                user_password,
                                0);


if (status != Security::SecAuthSuccess)
 {
    // 認証に失敗
    return;
 }
}
// アプリケーションに進む
...
// ログオフ
      try
         {
           logoff();
         }
...
```

Java Code Example That Uses the Tobj.PrincipalAuthenticator.logon() Method

リスト 10-6 には、`Tobj.PrincipalAuthenticator.logon()` メソッドを使用してパスワード認証を実行する Java コード例があります。

コード リスト 10-6 Tobj.PrincipalAuthenticator.logon() **メソッドを使用する**

Java クライアント・アプリケーション

```
...
      //Bootstrap オブジェクトを作成
      Tobj_Bootstrap bs =
         new Tobj_Bootstrap(orb, corbaloc://sling.com;2143);

      //SecurityCurrent オブジェクトを取得
      org.omg.CORBA.Object secCurObj =
         bs.resolve_initial_references( "SecurityCurrent" );
      org.omg.SecurityLevel2.Current secCur2Obj =
         org.omg.SecurityLevel2.CurrentHelper.narrow(secCurObj);

      //Principal Authenticator を取得
      org.omg.Security.PrincipalAuthenticator princAuth =
         secCur2Obj.principal_authenticator();
      com.beasys.Tobj.PrincipalAuthenticator auth =
         Tobj.PrincipalAuthenticatorHelper.narrow(princAuth);

      // 認証タイプを取得
      com.beasys.Tobj.AuthType authType = auth.get_auth_type();

      // 引数を初期化
      String userName = "John";
      String clientName = "Teller";
      String systemPassword = null;
      String userPassword = null;
      byte[] userData = new byte[0];

      // 要求されたセキュリティ・レベルに従って引数を準備
      switch(authType.value())
        {
        case com.beasys.Tobj.AuthType._TPNOAUTH:
           break;

        case com.beasys.Tobj.AuthType._TPSYSAUTH:
          systemPassword = "sys_pw";
          break;

        case com.beasys.Tobj.AuthType._TPAPPAUTH:
          systemPassword = "sys_pw";
          userPassword = "john_pw";
          break;
        }
```

```
      //Tuxedo 型の認証
      org.omg.Security.AuthenticationStatus status =
         auth.logon(userName, clientName, systemPassword,
                    userPassword, userData);
...
// アプリケーションに進む

// ログオフ
      try
         {
           auth.logoff();
         }
...
```

証明書による認証の使用

ここでは、CORBA アプリケーションでの証明書による認証のインプリメントについて説明します。

Secure Simpapp サンプル・アプリケーション

Secure Simpapp サンプル・アプリケーションは、既存の Simpapp サンプル・アプリケーションを使用し、SSL プロトコルおよび証明書による認証で保護された通信をサポートするようにコードおよびコンフィギュレーション・ファイルを変更します。

Secure Simpapp サンプル・アプリケーションのサーバ・アプリケーションは、以下の 2 つのメソッドを持つ CORBA オブジェクトのインプリメンテーションを提供します。

- `upper` メソッドは、クライアント・アプリケーションから文字列を受け取り、その文字列を大文字に変換します。
- `lower` メソッドは、クライアント・アプリケーションから文字列を受け取り、その文字列を小文字に変換します。

Simpapp サンプル・アプリケーションは、証明書による認証と SSL プロトコルをサポートするために以下のように変更されました。

- UBBCONFIG ファイルの ISL セクションで、IIOP リスナ / ハンドラに対して SSL プロトコルをコンフィギュレーションするために、ISL コマンドの -a、-S、-z、および -Z オプションが指定されています。
- UBBCONFIG ファイルの ISL セクションでは、IIOP リスナ / ハンドラに対する証明資料を指定するために、SEC_PRINCIPAL_NAME、SEC_PRINCIPAL_LOCATION、および SEC_PRINCIPAL_PASSVAR パラメータが定義されています。
- CORBA クライアント・アプリケーションのコードは、corbalocs URL アドレス形式を使用します。
- CORBA クライアント・アプリケーションのコードは、SecurityLevel2::PrincipalAuthenticator インターフェイスの authenticate() メソッドを使用し、プリンシパルを認証して、プリンシパルのクリデンシャルを取得します。

C++ Secure Simpapp サンプル・アプリケーションのソース・ファイルは、BEA Tuxedo ソフトウェアの \samples\corba\simpappSSL ディレクトリにあります。Secure Simpapp サンプル・アプリケーションの作成と実行の手順については、第 11 章の 1 ページ「CORBA サンプル・アプリケーションのビルドと実行」を参照してください。

CORBA クライアント・アプリケーションの記述

証明書による認証を使用する場合、以下を実行するクライアント・アプリケーション・コードを記述します。

1. Bootstrap オブジェクトを使用し、指定した BEA Tuxedo ドメインで SecurityCurrent オブジェクトのリファレンスを取得します。corbalocs URL アドレス形式を使用します。
2. SecurityCurrent オブジェクトから PrincipalAuthenticator オブジェクトを取得します。

3. SecurityLevel2::PrincipalAuthenticator インターフェイスの authenticate() メソッドを使用し、プリンシパルを認証して、プリンシパルのクリデンシャルを取得します。証明書による認証を使用する場合、method 引数に Tobj::CertificateBased を指定し、Security::Opaque の auth_data 引数として秘密鍵のパス・フレーズを指定します。

次の節には、証明書による認証のインプリメントを示す C++ および Java のコード例があります。

証明書による認証の C++ コード例

リスト 10-7 では、C++ CORBA クライアント・アプリケーションでの証明書による認証の使用を示します。

コード リスト 10-7 **証明書による認証を使用する** CORBA C++ **クライアント・アプリケーション**

```
....

// ORB の初期化
CORBA::ORB_var v_orb = CORBA::ORB_init(argc, argv, "");

//Bootstrap オブジェクトを作成
Tobj_Bootstrap bootstrap(v_orb.in(), corbalocs://sling.com:2143);

//SecurityCurrent を解決

CORBA::Object_ptr seccurobj =
        bootstrap.resolve_initial_references("SecurityCurrent");
SecurityLevel2::Current_ptr seccur =
        SecurityLevel2::Current::_narrow(seccurobj);

// 証明書ベースの認証を実行
        SecurityLevel2::Credentials_ptr   the_creds;
        Security::AttributeList_varprivileges;
        Security::Opaque_var continuation_data;
        Security::Opaque_var auth_specific_data;
        Security::Opaque_var response_data;

// プリンシパルの電子メール・アドレス
        char emailAddress[] = “milozzi@bigcompany.com;”
// プリンシパルのデジタル証明書のパス・フレーズ
        char password[] = “asdawrewe98infldi7;”
```

```
// 証明書プライベート・キー・パスワードをオペークに変換
        unsigned long password_len = strlen(password);
         Security::Opaque ssl_auth_data(password_len);

//Principal Authenticator でプリンシパルの証明書を認証
          for(int i = 0; (unsigned long) i < password_len; i++)
          ssl_auth_data[i] = password[i];
          Security::AuthenticationStatus auth_status;
          SecurityLevel2::PrincipalAuthenticator_var PA =
                           seccur->principal_authenticator();

          auth_status = PA->authenticate(Tobj::CertificateBased,
                                         emailAddress,
                                         ssl_auth_data,
                                         privileges,
                                         the_creds,
                                         continuation_data,
                                          auth_specific_data);

              while(auth_status == Security::SecAuthContinue) {
              auth_status = PA->continue_authentication(
                                         response_data,
                                         the_creds,
                                         continuation_data,
                                          auth_specific_data);

              }

...
```

証明書による認証の Java コード例

リスト 10-8 では、CORBA Java クライアント・アプリケーションでの証明書による認証の使用を示します。

コード リスト 10-8　**証明書による認証を使用する** CORBA Java **クライアント・アプリケーション**

```
...
//ORB を初期化
      Properties Prop;
```

```
      Prop = new Properties(System.getProperties());
      Prop.put("org.omg.CORBA.ORBClass","com.beasys.CORBA.iiop.ORB");
      Prop.put("org.omg.CORBA.ORBSingletonClass",
                "com.beasys.CORBA.idl.ORBSingleton");

      ORB orb = ORB.init(args, Prop);

//Bootstrap オブジェクトを作成

      Tobj_Bootstrap bs = new Tobj_Bootstrap(orb,
                corbalocs://foo:2501);

//SecurityCurrent を解決
      org.omg.CORBA.object ocurr =
         bs.resolve_initial_references("SecurityCurrent");
      org.omg.SecurityLevel2.Current curr =
         org.omg.SecurityLevel2.CurrentHelper.narrow(occur);

//Principal Authenticator を取得

      com.beasys.Tobj.PrincipalAuthenticator pa =
          (com.beasys.Tobj.PrincipalAuthenticator)
              curr.principal_authenticator();

      OpaqueHolder auth_data = new OpaqueHolder();
      AttributeListHolder privileges = new AttributeListHolder();
      org.omg.SecurityLevel2.CredentialsHolder creds =
          new org.omg.SecurityLevel2.CredentialsHolder();
      OpaqueHolder continuation_data = new OpaqueHolder();
      OpaqueHolder auth_specific_data = new OpaqueHolder();
      auth_data.value=new String ("deathstar").getbytes("UTF8);
      if(pa.authenticate(com.beasys.Tobj.CertificateBased.value,
                         "vader@largecompany.com",
                          auth_data.value,
                          privileges.value,
                          the_creds,
                          continuation_data,
                           auth_specific_data)

        !AuthenticationStatus.SecAuthSuccess) {
            System.err.println("logon failed");
            System.exit(1);
         }
       ...
```

インターオペラブル・ネーミング・サービス・メカニズムの使用

注記　このメカニズムは、サード・パーティのクライアント ORB で使用します。

インターオペラブル・ネーミング・サービス・メカニズムを使用し、適切なクリデンシャルで BEA Tuxedo ドメインにアクセスするには、次の手順を実行します。

1. `ORB::resolve_initial_references()` オペレーションを使用して、BEA Tuxedo ドメインの `SecurityLevel2::PrincipalAuthenticator` オブジェクトを取得します。`SecurityLevel2::PrincipalAuthenticator` オブジェクトは、固有の BEA 委譲型インターフェイスではなく、標準 CORBA サービス・セキュリティ・サービスに従い、認証プリンシパルを目的としたメソッドが含まれます。
2. `SecurityLevel2::PrincipalAuthenticator` オブジェクトの `authenticate()` メソッドを使用して、BEA Tuxedo ドメインにアクセスし、そのドメインに対してクライアント ORB を認証します。BEA Tuxedo ドメインへのアクセスにセキュリティ・クリデンシャルが必要な場合、`authenticate()` メソッドは、引き続き認証が必要であることを示す状態を返します。
3. `SecurityLevel2::PrincipalAuthenticator` インターフェイスの `continue_authentication()` メソッドを使用し、暗号化されたログオンとクリデンシャル情報を BEA Tuxedo ドメインに渡します。

CORBA インターオペラブル・ネーミング・サービス (INS) メカニズム使用の詳細については、「CORBA ブートストラップ処理のプログラミング・リファレンス」の `SecurityLevel2::PrincipalAuthenticator` インターフェイスの項目を参照してください。

invocations_options_required() メソッドの使用

証明書による認証を使用する場合、プリンシパルで必要なセキュリティ属性を定義しなければならない場合があります。たとえば、銀行取引アプリケーションでは、データをデータベースに転送する前に、特定のセキュリティ要件を満たさなければならない可能性があります。
`SecurityLevel2::Credentials` インターフェイスの `invocation_options_required()` メソッドを使用すると、プリンシパルは、SSL 接続のセキュリティ特性を明示的に制御できるようになります。
`corbaloc` URL アドレス形式を使用する場合、`SecurityLevel2::Credentials` インターフェイスの `authenticate()` メソッドと `invocation_options_required()` メソッドを使用し、ブートストラップ処理をセキュリティで保護できます。

`invocation_options_required()` メソッドを使用するには、次の手順を実行します。

1. `SecurityLevel2::PrincipalAuthenticator` オブジェクトの `authenticate()` メソッドを使用するアプリケーション・コードを記述し、証明書による認証が使用されるようにします。
2. `invocation_options_required()` メソッドを使用して、プリンシパルで必要なセキュリティ属性を指定します。セキュリティ・オプションのリストについては、第 15 章の 1 ページ「C++ セキュリティ・リファレンス」および第 16 章の 1 ページ「Java セキュリティ・リファレンス」の `invocation_options_required()` メソッドの項目を参照してください。

リスト 10-9 では、`invocation_options_required()` メソッドを使用した C++ の例を示します。

コード リスト 10-9 invocation_options_required() **メソッドを使用した** C++

の例

```
// ORB の初期化
CORBA::ORB_var v_orb = CORBA::ORB_init(argc, argv, "");

//Bootstrap オブジェクトを作成
Tobj_Bootstrap bootstrap(v_orb.in(), corbalocs://sling.com:2143);

//SecurityCurrent を解決

CORBA::Object_ptr seccurobj =
        bootstrap.resolve_initial_references("SecurityCurrent");
SecurityLevel2::Current_ptr seccur =
        SecurityLevel2::Current::_narrow(seccurobj);

// 証明書ベースの認証を実行
        SecurityLevel2::Credentials_ptr   the_creds;
Security::AttributeList_var        privileges;
        Security::Opaque_var continuation_data;
        Security::Opaque_var auth_specific_data;
        Security::Opaque_var response_data;

// プリンシパルの電子メール・アドレス
        char emailAddress[] = “milozzi@bigcompany.com;”
// プリンシパルのデジタル証明書のパス・フレーズ
        char password[] = “asdawrewe98infldi7;”

// 証明書プライベート・キー・パスワードをオペークに変換
        unsigned long password_len = strlen(password);
         Security::Opaque ssl_auth_data(password_len);

//Principal Authenticator でプリンシパルの証明書を認証
          for(int i = 0; (unsigned long) i < password_len; i++)
          ssl_auth_data[i] = password[i];
          Security::AuthenticationStatus auth_status;
          SecurityLevel2::PrincipalAuthenticator_var PA =
                          seccur->principal_authenticator();

          auth_status = PA->authenticate(Tobj::CertificateBased,
                                         emailAddress,
                                         ssl_auth_data,
                                         privileges,
                                         the_creds,
                                         continuation_data,
                                          auth_specific_data);
          the_creds->invocation_options_required(
                                Security::Integrity|
                                Security::DetectReplay|
```

```
                                        Security::DetectMisordering|
                                        Security::EstablishTrustInTarget|
                                        Security::EstalishTrustInClient|
                                        Security::SimpleDelegation);

                    while(auth_status == Security::SecAuthContinue) {
                          auth_status = PA->continue_authentication(
                                                response_data,
                                                the_creds,
                                                continuation_data,
                                                 auth_specific_data);

                        }
...
```

リスト 10-10 では、invocation_options_required() メソッドを使用した Java の例を示します。

コード リスト 10-10 invocation_options_required() **メソッドを使用した** Java **の例**

```
...
//ORB を初期化

      Properties Prop;
      Prop = new Properties(System.getProperties());
      Prop.put("org.omg.CORBA.ORBClass","com.beasys.CORBA.iiop.ORB");
      Prop.put("org.omg.CORBA.ORBSingletonClass",
                "com.beasys.CORBA.idl.ORBSingleton");

      ORB orb = ORB.init(args, Prop);

//Bootstrap オブジェクトを作成

      Tobj_Bootstrap bs = new Tobj_Bootstrap(orb,
                corbalocs://foo:2501);

//SecurityCurrent を解決
      org.omg.CORBA.object ocurr =
         bs.resolve_initial_references("SecurityCurrent");
      org.omg.SecurityLevel2.Current curr =
         org.omg.SecurityLevel2.CurrentHelper.narrow(occur);

//Principal Authenticator を取得
```

```
com.beasys.Tobj.PrincipalAuthenticator pa =
    (com.beasys.Tobj.PrincipalAuthenticator)
        curr.principal_authenticator();

OpaqueHolder auth_data = new OpaqueHolder();
AttributeListHolder privileges = new AttributeListHolder();
org.omg.SecurityLevel2.CredentialsHolder creds =
    new org.omg.SecurityLevel2.CredentialsHolder();
OpaqueHolder continuation_data = new OpaqueHolder();
OpaqueHolder auth_specific_data = new OpaqueHolder();
auth_data.value=new String ("deathstar").getbytes("UTF8);
if(pa.authenticate(com.beasys.Tobj.CertificateBased.value,
                   "vader@largecompany.com",
                    auth_data.value,
                    privileges.value,
                    the_creds,
                    continuation_data,
                     auth_specific_data)
org.omg.SecurityLevel2.Credentials credentials = curr.get_credentials(
    org.omg.Security.CredentialType.SecInvocationCredentials);

credentials.invocation_options_required(
    (short) (org.omg.Security.Integrity.value |
     org.omg.Security.DetectReplay.value|
     org.omg.Security.DetectMisordering.value|
     org.omg.Security.EstablishTrustInTarget.value|
     org.omg.Security.EstablishTrustInClient.value|
     org.omg.Security.SimpleDelegation.value)
    );
  !AuthenticationStatus.SecAuthSuccess) {
      System.err.println("logon failed");
      System.exit(1);
   }
 ...
```

章

11 CORBA サンプル・アプリケーションのビルドと実行

ここでは、次の内容について説明します。

- Security サンプル・アプリケーションのビルドと実行
- Secure Simpapp サンプル・アプリケーションのビルドと実行

Security サンプル・アプリケーションのビルドと実行

Security サンプル・アプリケーションでは、パスワード認証を使用します。Security サンプル・アプリケーションの作成と実行の手順については、『BEA Tuxedo CORBA University サンプル・アプリケーション』を参照してください。

Secure Simpapp サンプル・アプリケーションのビルドと実行

Secure Simpapp サンプル・アプリケーションでは、クライアント・アプリケーションと BEA Tuxedo ドメインの間の通信を保護するために SSL プロトコルと証明書認証を使用します。

Secure Simpapp サンプル・アプリケーションをビルドおよび実行するには、次の手順に従います。

1. Secure Simpapp サンプル・アプリケーションのファイルを作業ディレクトリにコピーします。
2. Secure Simpapp サンプル・アプリケーションのファイルに対する保護属性を変更します。
3. 環境変数を確認します。
4. `runme` コマンドを実行します。

Secure Simpapp サンプル・アプリケーションを使用する前に、IIOP リスナ / ハンドラの証明書と秘密鍵 (IIOPListener.pem) をユーザの企業の認証局から取得し、Lightweight Directory Access Protocol (LDAP) を有効にしたディレクトリ・サービスで証明書をロードします。runme コマンドで、IIOP リスナ / ハンドラの秘密鍵に対するパス・フレーズが表示されます。

ステップ 1: Secure Simpapp サンプル・アプリケーションのファイルを作業ディレクトリにコピーする

Secure Simpapp サンプル・アプリケーションのファイルを、ローカル・マシンの作業ディレクトリにコピーする必要があります。

Secure Simpapp サンプル・アプリケーションのファイルは、次のディレクトリにあります。

Windows 2000

*drive:**TUXdir*\samples\corba\simpappSSL

UNIX

/usr/local/TUXdir/samples/corba/simpappSSL

表 12-1 にリストされているファイルを使用して、Secure Simpapp サンプル・アプリケーションをビルドおよび実行します。

表 12-1 Secure Simpapp サンプル・アプリケーションに含まれるファイル

ファイル	説明
Simple.idl	Simple インターフェイスと SimpleFactory インターフェイスを宣言する OMG IDL コード。
Simples.cpp	デフォルトの Server::initialize メソッドと Server::release メソッドを上書きする C++ ソース・コード。

表 12-1 Secure Simpapp サンプル・アプリケーションに含まれるファイル (

ファイル	説明
`Simplec.cpp`	Secure Simpapp サンプル・アプリケーションの CORBA C++ クライアント・アプリケーションのソース・コード。
`Simple_i.cpp`	`Simple` メソッドと `SimpleFactory` メソッドをインプリメントする C++ ソース・コード。
`Simple_i.h`	`Simple` メソッドと `SimpleFactory` メソッドのインプリメンテーションを定義する C++ ヘッダ・ファイル。
`SimpleClient.java`	Secure Simpapp サンプル・アプリケーションのクライアント・アプリケーションの Java ソース・コード。
`Readme.html`	このファイルは、Secure Simpapp サンプル・アプリケーションのビルドと実行に関する最新の情報を提供します。
`runme.cmd`	Secure Simpapp サンプル・アプリケーションをビルドおよび実行する Windows 2000 バッチ・ファイル。
`runme.ksh`	Secure Simpapp サンプル・アプリケーションをビルドおよび実行する UNIX Korn シェル・スクリプト。
`makefile.mk`	UNIX オペレーティング・システムの Secure Simpapp サンプル・アプリケーションの makefile。このファイルは、Secure Simpapp サンプル・アプリケーションを手動でビルドするのに使用します。Secure Simpapp サンプル・アプリケーションを手動でビルドおよび実行する場合は、`Readme.html` ファイルを参照してください。UNIX の `make` コマンドは、ユーザのマシンのパス内に存在している必要があります。

表 12-1 Secure Simpapp サンプル・アプリケーションに含まれるファイル (

ファイル	説明
makefiles.nt	Windows 2000 オペレーティング・システムの Secure Simpapp サンプル・アプリケーションの makefile。この makefile は、Visual C++ の nmake コマンドで直接使用できます。このファイルは、Secure Simpapp サンプル・アプリケーションを手動でビルドするのに使用します。Secure Simpapp サンプル・アプリケーションを手動でビルドおよび実行する場合は、Readme.html ファイルを参照してください。Windows 2000 の nmake コマンドは、ユーザのマシンのパス内に存在している必要があります。

ステップ 2: Secure Simpapp サンプル・アプリケーションのファイルに対する保護属性を変更する

BEA Tuxedo ソフトウェアのインストール時には、サンプル・アプリケーションは読み取り専用に設定されています。Secure Simpapp サンプル・アプリケーションのファイルを編集またはビルドする前に、作業ディレクトリにコピーするファイルの保護属性を次のように変更する必要があります。

Windows 2000

```
prompt>attrib -r drive:\workdirectory\*.*
```

UNIX

```
prompt>/bin/ksh
ksh prompt>chmod u+w /workdirectory/*.*
```

また、UNIX オペレーティング・システム・プラットフォームでは、ファイルに実行権限を与えるために runme.ksh の権限を次のように変更する必要もあります。

```
ksh prompt>chmod +x runme.ksh
```

ステップ 3: 環境変数の設定を確認する

Secure Simpapp サンプル・アプリケーションをビルドおよび実行する前に、ユーザのシステムに対して一部の環境変数を設定する必要があります。ほとんどの場合、環境変数はインストール手順の一部として設定されます。ただし、環境変数をチェックして、正しい情報を反映していることを確認する必要があります。

表 12-2 では、Secure Simpapp サンプル・アプリケーションの実行に必要な環境変数を示します。

表 12-2 Secure Simpapp サンプル・アプリケーションで必須の環境変数

環境変数	説明
APPDIR	サンプル・アプリケーション・ファイルをコピーしたディレクトリ・パス。たとえば、次のように入力します。 Windows 2000 APPDIR=c:*work*\simpappSSL UNIX APPDIR=/usr/*work*/simpappSSL
TUXCONFIG	コンフィギュレーション・ファイルのディレクトリ・パスと名前。たとえば、次のように入力します。 Windows 2000 TUXCONFIG=c:\work\simpappSSL\tuxconfig UNIX TUXCONFIG=/usr/*work*/simpappSSL/tuxconfig
TOBJADDR	IIOP リスナ / ハンドラのホスト名とポート番号。ポート番号は、SSL 接続のポートとして定義する必要があります。たとえば、次のように入力します。 Windows 2000 TOBJADDR=trixie::1111 UNIX TOBJADDR=trixie::1111

表 12-2 Secure Simpapp サンプル・アプリケーションで必須の環境変数 (続き)

環境変数	説明
JAVA_HOME	JDK ソフトウェアをインストールしたディレクトリ・パス。たとえば、次のように入力します。 Windows 2000 JAVA_HOME=c:\JDK1.2 UNIX JAVA_HOME=/usr/local/JDK1.2 JAVA_HOME が定義されていない場合、サンプル・アプリケーションは、CORBA C++ クライアント・アプリケーションのみを使用します。
RESULTSDIR	runme コマンドの実行により作成されたファイルが格納されている APPDIR のサブディレクトリ。たとえば、次のように入力します。 Windows 2000 RESULTSDIR=c:\workdirectory\ UNIX RESULTSDIR=/usr/local/workdirectory/

インストール中に定義された環境変数の情報が正しいことを確認するには、以下の手順に従います。

Windows 2000

1. [スタート] メニューの、[設定] をポイントします。
2. [設定] メニューから、[コントロール パネル] をクリックします。

   [コントロール パネル] が表示されます。
3. [システム] アイコンをクリックします。

   [システムのプロパティ] ウィンドウが表示されます。
4. [詳細] タブの [環境変数] をクリックします。

   [環境変数] ページが表示されます。
5. 環境変数の設定をチェックします。

UNIX

ksh prompt>printenv TUXDIR

ksh prompt>printenv JAVA_HOME (CORBA Java クライアント・アプリケーションの場合)

設定を変更するには、以下の手順に従います。

Windows 2000

1. [システムのプロパティ] ウィンドウの [環境] ページで、変更する環境変数をクリックするか、[変数] フィールドに環境変数の名前を入力します。
2. [値] フィールドに、環境変数の正しい情報を入力します。
3. [OK] をクリックして変更を保存します。

UNIX

ksh prompt>export TUXDIR=*directorypath*

ksh prompt>export JAVA_HOME=*directorypath* (CORBA Java クライアント・アプリケーションの場合)

ステップ 4: runme コマンドを実行する

runme コマンドは、以下の手順を自動化します。

1. システム環境変数の設定
2. UBBCONFIG ファイルのロード
3. クライアント・アプリケーションのコードのコンパイル
4. クライアント・アプリケーションのコードのコンパイル
5. tmboot コマンドによるサーバ・アプリケーションの起動
6. クライアント・アプリケーションの起動
7. tmshutdown コマンドによるサーバ・アプリケーションの終了

注記　Secure Simpapp サンプル・アプリケーションを手動で実行することもできます。Secure Simpapp サンプル・アプリケーションを手動で実行する手順については、Readme.html ファイルで説明しています。

Secure Simpapp サンプル・アプリケーションをビルドおよび実行するには、次のように runme コマンドを入力します。

Windows 2000

```
prompt>cd workdirectory

prompt>runme
```

UNIX

```
ksh prompt>cd workdirectory

ksh prompt>./runme.ksh
```

Secure Simpapp サンプル・アプリケーションが起動し、次のメッセージが表示されます。

```
Testing simpapp
    cleaned up
    prepared
    built
    loaded ubb
    booted
    ran
    shutdown
    saved results
  PASSED
```

runme コマンドの実行時に、パスワードの入力を要求されます。IIOP リスナ / ハンドラの秘密鍵のパス・フレーズを入力します。

表 12-3 では、runme コマンドで作業ディレクトリ内に生成された C++ ファイルを示しています。

表 12-3 runme コマンドで生成される C++ ファイル

ファイル	説明
Simple_c.cpp	idl コマンドによって生成されます。このファイルは、SimpleFactory インターフェイスと Simple インターフェイスのクライアント・スタブを格納します。
Simple_c.h	idl コマンドによって生成されます。このファイルは、SimpleFactory インターフェイスと Simple インターフェイスのクライアント定義を格納します。
Simple_s.cpp	idl コマンドによって生成されます。このファイルは、SimpleFactory インターフェイスと Simple インターフェイスのサーバ・スケルトンを格納します。
Simple_s.h	idl コマンドによって生成されます。このファイルは、SimpleFactory インターフェイスと Simple インターフェイスのサーバ定義を格納します。

表 12-4 では、runme コマンドで作業ディレクトリ内に生成された Java ファイルを示しています。

表 12-4 runme コマンドで生成される Java ファイル

ファイル	説明
SimpleFactory.java	SimpleFactory インターフェイス用に idltojava コマンドによって生成されます。SimpleFactory インターフェイスには、Java バージョンの OMG IDL インターフェイスが含まれます。これは、org.omg.CORBA.Object の拡張です。

表 12-4 runme コマンドで生成される Java ファイル (続き)

ファイル	説明
SimpleFactoryHolder.java	SimpleFactory インターフェイス用に idltojava コマンドによって生成されます。このクラスは、SimpleFactory 型のパブリック・インスタンス・メンバを保持します。また、CORBA に含まれているものの、Java に正確にマップされない out 引数と inout 引数に対するオペレーションを提供します。
SimpleFactoryHelper.java	SimpleFactory インターフェイス用に idltojava コマンドによって生成されます。このクラスは、補助機能、特に narrow メソッドを提供します。
_SimpleFactoryStub.java	SimpleFactory インターフェイス用に idltojava コマンドによって生成されます。このクラスは、SimpleFactory.java インターフェイスをインプリメントするクライアント・スタブです。
Simple.java	Simple インターフェイス用に idltojava コマンドによって生成されます。Simple インターフェイスには、Java バージョンの OMG IDL インターフェイスが含まれます。これは、org.omg.CORBA.Object の拡張です。
SimpleHolder.java	Simple インターフェイス用に idltojava コマンドによって生成されます。このクラスは、Simple 型のパブリック・インスタンス・メンバを保持します。また、CORBA に含まれているものの、Java に正確にマップされない out 引数と inout 引数に対するオペレーションを提供します。

表 12-4 runme コマンドで生成される Java ファイル (続き)

ファイル	説明
SimpleHelper.java	Simple インターフェイス用に idltojava コマンドによって生成されます。このクラスは、補助機能、特に narrow メソッドを提供します。
_SimpleStub.java	Simple インターフェイス用に idltojava コマンドによって生成されます。このクラスは、Simple.java インターフェイスをインプリメントするクライアント・スタブです。

表 12-5 では、runme コマンドで RESULTS ディレクトリ内に生成されるファイルを示しています。

表 12-5 runme コマンドで results ディレクトリ内に生成されるファイル

ファイル	説明
input	runme コマンドが CORBA クライアント・アプリケーションに提供する入力を格納します。
output	runme コマンドが CORBA クライアント・アプリケーションを実行するときに生成される出力を格納します。
expected_output	runme コマンドが CORBA クライアント・アプリケーションを実行するときに予測される出力を格納します。テストに成功したか失敗したかを判別するために、output ファイルのデータは expected_output ファイルのデータと比較されます。
log	runme コマンドで生成される出力を格納します。runme コマンドが失敗すると、このファイルのエラーをチェックします。

表 12-5 runme コマンドで results ディレクトリ内に生成されるファイル (続

ファイル	説明
setenv.cmd	Windows 2000 オペレーティング・システム・プラットフォームの Secure Simpapp サンプル・アプリケーションのビルドと実行に必要な環境変数を設定するためのコマンドを格納します。
stderr	tmboot コマンドによって生成されます。このコマンドは、runme コマンドによって実行されます。
stdout	tmboot コマンドによって生成されます。このコマンドは、runme コマンドによって実行されます。
tmsysevt.dat	TMSYSEVT (システム・イベント・リポート) プロセスで使用するフィルタ規則および通知規則を格納します。このファイルは、runme コマンドの tmboot コマンドによって生成されます。
tuxconfig	バイナリ形式の UBBCONFIG ファイル。
ULOG.<date>	tmboot コマンドによって生成されるメッセージを含んだログ・ファイル。

Secure Simpapp サンプル・アプリケーションの使用

Secure Simpapp サンプル・アプリケーションのサーバ・アプリケーションを次のように実行します。

Windows 2000

```
prompt>tmboot -y
```

UNIX

```
ksh prompt>tmboot -y
```

Secure Simpapp サンプル・アプリケーションの CORBA C++ クライアント・アプリケーションを次のように実行します。

Windows 2000

```
prompt> set TOBJADDR=corbalocs://host:port
prompt> simple_client -ORBid BEA_IIOP -ORBpeerValidate none
String?
Hello World
HELLO WORLD
hello world
```

UNIX

```
ksh prompt>export TOBJADDR=corbalocs://host:port
ksh prompt>simple_client -ORBid BEA_IIOP -ORBpeerValidate none
String?
Hello World
HELLO WORLD
hello world
```

Secure Simpapp サンプル・アプリケーションの CORBA Java クライアント・アプリケーションを次のように実行します。

Windows 2000

```
prompt> set CLASSPATH=%TUXDIR%\udataobj\java\jdk\m3envobj.jar;
%TUXDIR%\udataobj\java\jdk\wleclient.jar;.;%CLASSPATH%
java -DTOBJADDR=%TOBJADDR% -Dorg.omg.CORBA.ORBpeerValidate=none
classpath %CLASSPATH% SimpleClient
String?
Hello World
HELLO WORLD
hello world
```

UNIX

```
ksh prompt>export
CLASSPATH=${TUXDIR}/udataobj/java/jdk/m3envobj.jar;
${TUXDIR}/udataobj/java/jdk/wleclient.jar:.:${CLASSPATH}
java -DTOBJADDR=${TOBJADDR} -Dorg.omg.CORBA.ORBpeerValidate=none
-classpath ${CLASSPATH} SimpleClient
String?
Hello World
HELLO WORLD
hello world
```

注記　Secure Simpapp サンプル CORBA Java クライアント・アプリケーションの CORBA Java クライアント・アプリケーションは、クライアントのみの JAR ファイル `m3envobj.jar` および `wleclient.jar` を使用します。

別のサンプル・アプリケーションを使用する前に、以下のコマンドを入力して Secure Simpapp サンプル・アプリケーションを終了し、不要なファイルを作業ディレクトリから削除します。

Windows 2000

```
prompt>tmshutdown -y

prompt>nmake -f makefile.nt clean
```

UNIX

```
ksh prompt>tmshutdown -y

ksh prompt>make -f makefile.mk clean
```

章

12 トラブルシューティング

ここでは、以下の内容について説明します。

- ULOG および ORB トレース機能の使用
- CORBA::ORB_init Problems
- パスワード認証の問題
- 証明書による認証の問題
- Tobj::Bootstrap:: resolve_initial_references の問題
- IIOP リスナ / ハンドラの起動の問題
- コンフィギュレーションの問題
- SSL プロトコルでコールバック・オブジェクトを使用する場合の問題
- デジタル証明書のトラブルシューティングのヒント

注記　ここで述べられている問題は、CORBA アプリケーションで SSL プロトコルと証明書による認証を使用している場合に当てはまります。

ULOG および ORB トレース機能の使用

通常、オブジェクト・リクエスト・ブローカ (ORB) は、重大な障害を ULOG ファイルに書き込みます。CORBA C++ ORB では、ORB 内部トレース機能を有効にすることで、ULOG ファイルの情報以外の情報を参照することもできます。

ULOG ファイルを見ると、リモート ORB プロセスはデフォルトでは、データを APPDIR の ULOG ファイルには書き込みません。

- UNIX では、リモート ORB は、情報をカレント・ディレクトリの ULOG ファイルに書き込みます。
- Windows 2000 では、リモート ORB は、情報を c:\ulog ディレクトリの ULOG ファイルに書き込みます。

ULOGPFX 環境変数を設定すると、リモート ORB 用の ULOG ファイルの場所を管理できます (たとえば、ULOG ファイルの場所を APPDIR に設定し、すべての情報が同じ ULOG ファイルに格納されるようにすることができます)。ULOGPFX 環境変数を次のように設定します。

Windows 2000

```
set ULOGPFX=%APPDIR%\ULOG
```

UNIX

```
setenv ULOGPFX $APPDIR/ULOG
```

ORB トレース機能を有効にするには、次の手順を実行します。

1. APPDIR に trace.dat という名前のファイルを作成します。trace.dat 内で、all=on が指定されている必要があります。
2. 以下のコマンドを使用して OBB_TRACE_INPUT 環境変数を設定し、trace.dat ファイルを参照してからアプリケーションを実行するようにします。

   ```
   set OBB_TRACE_INPUT=%APPDIR%\trace.dat
   ```

ORB トレース機能を別のファイルに送信する場合、次の行を trace.dat ファイルに追加します。

```
output=obbtrace%p.log
```

このコマンドは、トレース出力を、実行中のプロセスにちなんだ名前のファイルに送信します。たとえば、UNIX の ORB トレース機能を、NFS がマウントされたドライブに対して使用する場合に実行できます。この場合、トレース文ごとにユーザ・ログがファイルを開き、書き込み、閉じるために、トレースの性能は遅くなります。

CORBA::ORB_init Problems

ORB_init ルーチンは、内部 ORB トレース機能を実行しないので、無効な引数の処理に対するトレース出力を確認できません。したがって、ORB_init ルーチンに渡された引数を二重チェックする必要があります。

ORB_init ルーチンの実行中に CORBA::BAD_PARAM 例外が発生した場合は、必要なすべての引数に値が指定されていることを確認してください。また、特定の有効な値のセットからの値を予測している引数に正しい値があることも確認してください。ORB_init ルーチンの引数の値では、大文字と小文字を区別するので注意してください。

CORBA::NO_PERMISSION 例外が発生し、SSL 引数が ORB_init ルーチンに指定された場合、セキュリティ・ライセンスが有効になっていることを確認してください。また、指定した暗号化レベルが、セキュリティ・ライセンスでサポートされている暗号化レベルを超えていないことを確認してください。

ORB_init ルーチンの実行中に CORBA::IMP_LIMIT 例外が発生した場合は、システム・プロパティの ORBport と ORBSecurePort に同じ値が指定されていることを確認してください。

ORB_init ルーチンの実行中に CORBA::Initialize 例外が発生した場合は、OrbId または configset が有効であることを確認してください。

セキュア・ソケット・レイヤ (SSL) 引数が ORB_init ルーチンに渡されている場合、ORB は、SSL プロトコルをロードおよび初期化しようとします。SSL 引数が渡された場合、ORB は、SSL プロトコルを初期化しようとします。

ORB は、Bootstrap オブジェクトの新しい URL アドレス形式を認識しないので、corbaloc または corbalocs URL アドレス形式を指定した場合、ORB は、ORB_init ルーチンの実行時に SSL プロトコルのロードを試行しません。

SSL 引数を ORB_init ルーチンに対して指定した場合は、以下をチェックします。

- SSL 引数に対して指定した値が、互いに競合しないこと、またはほかの ORB 引数と競合しないこと。
- ORB がネイティブ・プロセスかどうか。ORB がネイティブ・プロセスの場合、SSL 引数はサポートされません。
- maxCrypto システム・プロパティ用に指定された値が、minCrypto システム・プロパティ用に指定された値より低いこと。この 2 つのプロパティの値は、ライセンスに適した範囲内にある必要があります。
- アプリケーション制御の不正な SSL コンフィギュレーション・パラメータ。ORB_init ルーチンは、デジタル証明書のルックアップを実行しないので、ダイナミック・ライブラリがロードされない、不明のファイルまたは障害が発生したファイルを探してください。また、ダイナミック・ライブラリがロードされていることも確認してください。ORB トレース機能は、ダイナミック・ライブラリがロードされたかどうかに関する情報を提供します。

問題が続く場合は、ORB トレース機能を有効にします。ORB トレース機能は、liborbssl ダイナミック・ライブラリがロードされて初期化されたときに発生する SSL の障害をログに記録します。

パスワード認証の問題

パスワード認証に corbalocs URL アドレス形式を使用しているときにクライアント・アプリケーションが失敗した場合は、以下をチェックします。

- コンフィギュレーションの手順が正しく実行されたかどうか。必要なコンフィギュレーション手順のリストについては、「SSL プロトコルのコンフィギュレーション」および「認証のコンフィギュレーション」を参照してください。
- 初期化エラーが発生したかどうか。SSL システム・プロパティを ORB_init ルーチンに対して指定すると、以下の場合にエラーが発生します。
  - IIOP リスナ / ハンドラが使用不可能な場合。ORB トレース・ログには、失敗した接続試行が記録されます。
  - IIOP リスナ / ハンドラは使用可能だが、SSL プロトコルをサポートしていない場合。ULOG ファイルには、非 GIOP メッセージを受信したことが記録されます。
  - IIOP リスナ / ハンドラが使用可能で、SSL プロトコル用にコンフィギュレーションされているが、SSL 接続を確立できなかった場合。IIOP リスナ / ハンドラがサポートしている暗号化レベルの範囲と、クライアント・アプリケーションで必要な暗号化レベルの範囲が一致していない場合に、このエラーが発生します。

    IIOP リスナ / ハンドラが SSL プロトコル用にコンフィギュレーションされているものの、セキュリティで保護されている接続を示す corbalocs 接頭辞なしに CORBA クライアント・アプリケーションが TOBJADDR オブジェクトを使用している場合に、ULOG ファイルに、非 GIOP メッセージを受信したことが記録されます。

証明書による認証の問題

証明書による認証に corbalocs URL アドレス形式を使用しているときにクライアント・アプリケーションが失敗した場合は、以下をチェックします。

- コンフィギュレーションの手順が正しく実行されたかどうか。必要なコンフィギュレーション手順のリストについては、第 6 章の 1 ページ「SSL プロトコルのコンフィギュレーション」および第 7 章の 1 ページ「認証のコンフィギュレーション」を参照してください。
- 初期化エラーが発生したかどうか。
- SSL システム・プロパティを ORB_init ルーチンに対して指定すると、以下の場合にエラーが発生します。
  - IIOP リスナ / ハンドラが使用不可能な場合。ORB トレース・ログには、失敗した接続試行が記録されます。
  - IIOP リスナ / ハンドラは使用可能だが、SSL プロトコルをサポートしていない場合。ULOG ファイルには、非 GIOP メッセージを受信したことが記録されます。
  - IIOP リスナ / ハンドラが使用可能で、SSL プロトコル用にコンフィギュレーションされているが、SSL 接続を確立できなかった場合。IIOP リスナ / ハンドラがサポートしている暗号化レベルの範囲と、クライアント・アプリケーションで必要な暗号化レベルの範囲が一致していない場合に、このエラーが発生します。また、クライアント・アプリケーションが、IIOP リスナ / ハンドラの証明書チェーンを信頼しなかった場合、またはクライアント・アプリケーションが IIOP リスナ / ハンドラから証明書を受け取らなかった場合にも、エラーが発生します。エラーは、ULOG ファイルに書き込まれ、ORB トレース出力にも表示されます。

  エラーが発生しない場合、問題は認証プロセスにあります。ULOG ファイルには、問題を示す以下のエラー文のいずれかが含まれています。
  - `Couldn’t connect to an LDAP server`
  - `Couldn’t find a filter that matched the client certificate`
  - `The client certificate was not found in LDAP`

- The private key file could not be found
- The passphrase used to open the private key is not correct
- The public key from the client certificate did not match the private key

ほかの認証に関する問題が発生する場合もあります。発生の可能性がある認証エラーの種類については、第 12 章の 7 ページ「Tobj::Bootstrap:: resolve_initial_references の問題」を参照してください。

注記　初期化プロセスの時点で発生した障害は、IIOP リスナ / ハンドラの問題ではありません。

Tobj::Bootstrap:: resolve_initial_references の問題

corbaloc または corbalocs URL アドレス形式で Tobj::Bootstrap::resolve_initial_references を実行しているときに障害が発生した場合、CORBA::InvalidDomain 例外が発生します。この例外は、内部で発生する CORBA::NO_PERMISSION 例外または CORBA::COMM_FAILURE 例外をマスクする場合があります。ULOG ファイルを参照し、ORB トレース機能を有効にしてエラーの詳細を確認してください。以下のエラーが発生する可能性があります。

- IIOP リスナ / ハンドラが使用不可能な場合、ORB トレース・ログには、失敗した接続試行が記録されます。
- IIOP リスナ / ハンドラは使用可能だが、SSL プロトコルをサポートしていない場合、ULOG ファイルには、非 GIOP メッセージを受信したことが示されます。
- IIOP リスナ / ハンドラが使用可能で、SSL プロトコル用にコンフィギュレーションされているが、SSL 接続を確立できなかった場合。IIOP リスナ / ハンドラがサポートしている暗号化レベルの範囲と、クライアント・アプリケーションで必要な暗号化レベルの範囲が一致していない場合に、このエラーが発生します。

- IIOP リスナ / ハンドラが、証明書をユーザ名 / パスワードの組み合わせにマップできなかった場合。CORBA アプリケーションのセキュリティのレベルが USER_AUTH に設定されていること、および指定したユーザ名が、認証呼び出しに渡されたプリンシパル名に一致していることを確認します。また、ユーザ名が 30 文字を超えていないこともチェックしてください。

ほかの認証に関する問題が発生する場合もあります。発生の可能性がある認証エラーの種類については、第 12 章の 11 ページ「デジタル証明書のトラブルシューティングのヒント」を参照してください。

注記　Tobj_Bootstrap::resolve_initial_references() メソッドの Java インプリメンテーションは、InvalidDomain 例外をスローしません。corbaloc または corbalocs URL アドレス形式を使用している場合、Tobj_Bootstrap::resolve_initial_references() メソッドは、内部で InvalidDomain 例外をキャッチし、COMM_FAILURE を例外としてスローします。このメソッドは、下位互換性を提供するためにこのように機能します。

IIOP リスナ / ハンドラの起動の問題

ここでは、IIOP リスナ / ハンドラの起動時に発生する可能性がある問題について説明します。

IIOP リスナ / ハンドラを起動しているときに障害が発生した場合、ULOG ファイルをチェックしてエラーの詳細を確認します。IIOP リスナ / ハンドラは、CLOPT パラメータで指定した SSL 引数の値が有効かどうかを確認します。値のいずれかが無効な場合、対応するエラーが ULOG ファイルに記録されます。このチェックは、ORB で実行される引数チェックに似ています。

IIOP リスナ / ハンドラは、-m オプションが指定されない限りプロセスを開始しません。ISH は、SSL ライブラリを実際にロードおよび初期化するプロセスです。ISH プロセスの SSL ライブラリのロードおよび初期化で問題が発生した場合、ISH プロセスがクライアント・アプリケーションの要求の処理を開始するまで、エラーは ULOG ファイルに記録されます。

IIOP リスナ / ハンドラ・プロセスの起動で問題が発生したと思われる場合は、ULOG ファイルを確認してください。

コンフィギュレーションの問題

ここでは、セキュリティ使用時に発生する一般的なコンフィギュレーションの問題を解決する方法について説明します。

- ISL コマンドの ORB -ORBpeerValidate コマンド行オプションと -v オプションは、ピア検証の規則のチェックを制御しません。このシステム・プロパティとオプションは、プリンシパルの接続先マシンのホスト名に対するピア証明書で指定されたホスト名のチェックのみを制御します。
- インストールしたキットでピア検証の規則を無効にする唯一の方法は、%TUXDIR%\udataobj\security\certs\peer_val.rul に対して空のファイルを作成することです。CORBA アプリケーションを構築するスクリプトを記述する場合、スクリプトで peer_val.rul ファイルを登録できません。
- IIOP リスナ / ハンドラで再調整の間隔を有効にしている場合、ISL コマンドのオプションが -r ではなく -R であることを確認します。-r を使用する場合、IIOP リスナ / ハンドラは SSL プロトコルを使用しますが、再調整の間隔は使用されません。また、ULOG ファイルには、IIOP リスナ / ハンドラで不明なオプションを指定したことが記録されます。

  IIOP リスナ / ハンドラが再調整を実行しているかどうかを判定する別の方法は、クライアント側で ORB トレース機能を有効にし、暗号スイート調整コールバックがコンフィギュレーション済みの再調整の間隔を呼び出しているかどうかをチェックすることです。再調整が発生するためには、クライアント・アプリケーションが要求を送信している必要があります。
- CORBA アプリケーションの UBBCONFIG ファイルで SECURITY パラメータを APP_PW 以上になるように定義しており、SSL プロトコルを使用するように (ただし相互認証ではなく) IIOP リスナ / ハンドラをコンフィギュレーションしている場合、IIOP リスナ / ハンドラと通信するために

corbalocs URL アドレス形式でパスワード認証を使用する必要があります。証明書による認証を使用しようとしても、IIOP リスナ / ハンドラは、SSL 接続を確立するときにプリンシパルに証明書を要求しないので、IIOP リスナ / ハンドラは、プリンシパルの ID を BEA Tuxedo の ID にマップできません。

SSL プロトコルでコールバック・オブジェクトを使用する場合の問題

共同クライアント / サーバ・アプリケーションがあり、そのクライアント側で corbalocs URL アドレス形式を使用するか、クリデンシャルを要求してセキュリティ要件を指定している場合、ORB_init ルーチンで -ORBsecurePort システム・プロパティを使用して、使用するセキュリティ保護ポートを指定する必要があります。

-ORBsecurePort システム・プロパティを指定しない場合、CORBA::NO_PERMISSION 例外が発生してサーバ登録が失敗します。これが問題かどうかを検証するには、ORB トレース機能を有効にして、以下のトレース出力を探します。

```
TCPTransport::Listen: FAILURE: Attempt to listen on clear port
while Credentials require SSL be used
```

コールバック・オブジェクトで SSL プロトコルを使用する場合、共同クライアント / サーバ・アプリケーションは、証明書による認証で SecurityLevel2::PrincipalAuthenticator::authenticate() メソッドを使用する必要があります。それ以外の場合、共同クライアント / サーバ・アプリケーションは、IIOP リスナ / ハンドラ (ここでは、SSL 接続のイニシエータ) に対して自身を識別するための証明書を持ちません。

デジタル証明書のトラブルシューティングのヒント

通常、デジタル証明書の問題は、以下の場合に起こります。

- IIOP リスナ/ハンドラの証明書チェーンにあるデジタル証明書のいずれか 1 つが `trust_ca.cer` ファイルで定義した認証局から受け取ったものではない場合。`trust_ca.cer` ファイルの認証局が無効な場合に、問題が発生することがあります。
- クライアント・アプリケーションに接続された IIOP リスナ / ハンドラが、ホスト一致チェックを実行したときに IIOP リスナ / ハンドラのデジタル証明書で指定したホスト名と同じでない場合。IIOP リスナ / ハンドラ名は、IIOP リスナ / ハンドラの固有名の `CommonName` 属性で指定します。ホスト名と `CommonName` 属性は、正確に一致している必要があります。

  `-ORBpeerValidate` システム・プロパティを `none` に設定し、再度 `ORB_init` ルーチンを実行すると、このエラーを検証できます。
- IIOP リスナ/ハンドラの証明書チェーンにあるデジタル証明書のいずれか 1 つが、指定したピア検証の規則と一致していない場合。
- IIOP リスナ / ハンドラのデジタル証明書が無効の場合。IIOP リスナ / ハンドラのデジタル証明書は、改ざんされた場合、期限切れの場合、または発行元の認証局が期限切れの場合、無効になります。

理由がわからずにデジタル証明書が拒否された場合は、次の手順を実行します。

1. Microsoft Explorer などのビューアでデジタル証明書を開きます。
2. デジタル証明書の `KeyUsage` プロパティおよび `BasicConstraints` プロパティを確認します。感嘆符が付いた黄色い三角形のマークは、そのプロパティに問題があることを示します。プロパティにそのマークが付いたデジタル証明書は、BEA Tuxedo ソフトウェアによって拒否されます。

3. デジタル証明書のそのプロパティに問題がない場合、証明書チェーン内の次のデジタル証明書のプロパティをチェックします。証明書チェーン内のすべてのデジタル証明書に対して、そのプロパティを検証するまで、以上の手順を繰り返します。

第 IV 部セキュリティ・リファレンス

章

13 CORBA セキュリティ API

ここでは、以下の内容について説明します。

- CORBA セキュリティ・モデル
- CORBA セキュリティ環境で機能するコンポーネント
- Principal Authenticator オブジェクト
- Credentials オブジェクト
- SecurityCurrent オブジェクト

CORBA セキュリティ API の C++、Java、およびオートメーション・メソッドの説明については、以下の項目を参照してください。

- 第 15 章の 1 ページ「C++ セキュリティ・リファレンス」
- 第 16 章の 1 ページ「Java セキュリティ・リファレンス」
- 第 17 章の 1 ページ「オートメーション・セキュリティ・リファレンス」

CORBA セキュリティ・モデル

BEA Tuxedo 製品の CORBA 環境のセキュリティ・モデルは、セキュリティのフレームワークのみを定義します。BEA Tuxedo 製品は、異なるセキュリティ・メカニズムをサポートする柔軟性と、特定の CORBA アプリケーションに対応したレベルの機能と確実性の実現に使用できる方針を提供します。

CORBA 環境のセキュリティ・モデルは、以下を定義します。

- クライアント・アプリケーションが、BEA Tuxedo ドメインでオブジェクトにアクセスできるようになるための条件
- プリンシパルが BEA Tuxedo ドメインに対して自身を認証するのに必要な証明資料のタイプ

CORBA 環境のセキュリティ・モデルは、CORBA サービス・セキュリティ・サービス仕様で定義したセキュリティ・モデルと、BEA Tuxedo 製品の ATMI 環境にある、機能を特化して単純化したセキュリティ・モデルを提供する、値を追加した拡張の組み合わせです。

以下の節では、CORBA セキュリティ・モデルの一般的な特性を説明します。

プリンシパルの認証

プリンシパル (個々のユーザ、クライアント・アプリケーション、サーバ・アプリケーション、共同クライアント / サーバ・アプリケーション、または IIOP リスナ / ハンドラなど) の認証を使用すると、セキュリティ担当者は、登録したプリンシパルのみがシステムのオブジェクトにアクセスできるようにすることができます。認証されたプリンシパルは、オブジェクトへのアクセスを制御する主要なメカニズムとして使用します。プリンシパルの認証を使用すると、セキュリティ・メカニズムで以下のことが可能になります。。

- プリンシパルをそれぞれのアクションに対応したものにすることができます。

- 保護されたオブジェクトへのアクセスを制御できます。
- 要求の発信元を識別できます。
- 要求の発信先を識別できます。

オブジェクトへのアクセスの制御

CORBA セキュリティ・モデルは、セキュリティ担当者が、BEA Tuxedo ドメインに対するアクセスを権限のあるユーザのみに制限できる単純なフレームワークを提供します。オブジェクトへのアクセスを制限すると、セキュリティ担当者は、権限のないプリンシパルによるオブジェクトへのアクセスを禁止できます。アクセス制御フレームワークは、2 つの部分で構成されています。

- オブジェクト呼び出しに対して自動的に適用されるオブジェクト呼び出し方針
- ユーザが記述したアプリケーションが使用するアプリケーション・アクセス方針

Administrative Control

システム管理者には、CORBA アプリケーションに対するセキュリティ方針の設定する役割があります。BEA Tuxedo 製品は、一連のコンフィギュレーション・パラメータとユーティリティを提供します。コンフィギュレーション・パラメータとユーティリティを使用すると、システム管理者は、プリンシパルを認証し、BEA Tuxedo ソフトウェアがインストールされているシステムにそのプリンシパルがアクセスできるように CORBA アプリケーションをコンフィギュレーションできます。コンフィギュレーション・パラメータを設定するには、システム管理者は、`tmloadcf` コマンドを使用し、特定の CORBA アプリケーションのコンフィギュレーション・ファイルを更新します。

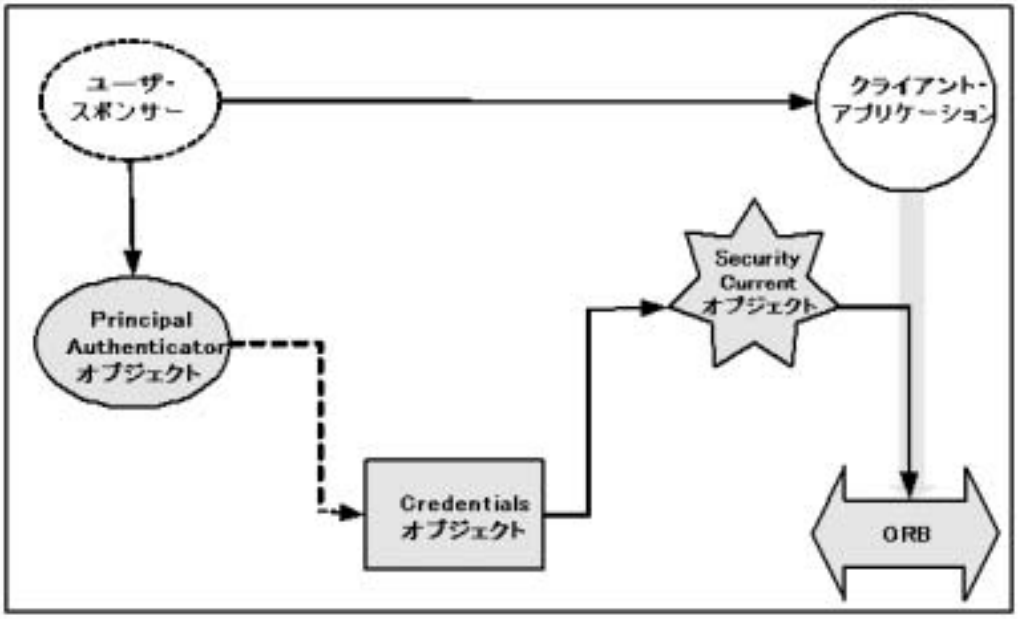
CORBA アプリケーションのセキュリティのコンフィギュレーションについては、第 6 章の 1 ページ「SSL プロトコルのコンフィギュレーション」および第 7 章の 1 ページ「認証のコンフィギュレーション」を参照してください。

CORBA セキュリティ環境で機能するコンポーネント

CORBA セキュリティ・モデルは、BEA Tuxedo ドメインに対してプリンシパルを認証するプロセスを基にしています。CORBA セキュリティ環境のオブジェクトは、プリンシパルの認証に使用します。プリンシパルは、パスワードなどの ID および認証データをクライアント・アプリケーションに提供します。クライアント・アプリケーションは、Principal Authenticator オブジェクトを使用して、プリンシパルの認証に必要な呼び出しを実行します。認証されたプリンシパルのクリデンシャルは、セキュリティ・システムの SecurityCurrent オブジェクトのインプリメンテーションと関連付けられ、Credentials オブジェクトで表されます。

図 13-1 では、CORBA セキュリティ・モデルで使用する認証プロセスを示しています。

図 13-1CORBA セキュリティ・モデルでの認証プロセス



以下の節では、CORBA セキュリティ・モデルのオブジェクトを説明します。

Principal Authenticator オブジェクト

Principal Authenticator オブジェクトは、認証を受ける必要があって、まだ受けていないプリンシパルで、オブジェクト・システムを呼び出す前に使用します。プリンシパルの認証によって、Credentials オブジェクトが作成されます。これは、アプリケーションのデフォルト・クリデンシャルとして使用できます。

Principal Authenticator オブジェクトは、シングルトン・オブジェクトです。つまり、プロセス・アドレス領域で許可されるインスタンスは 1 つだけです。また、Principal Authenticator は、状態を持たないオブジェクトです。Credentials オブジェクトは、作成元の Principal Authenticator オブジェクトには関連付けられていません。

すべての Principal Authenticator オブジェクトは、CORBA サービス・セキュリティ・サービス仕様で定義されている `SecurityLevel2::PrincipalAuthenticator` インターフェイスをサポートします。このインターフェイスには、プリンシパルの認証に使用する 2 つのメソッドがあります。メソッドが 2 つあるのは、プリンシパルの認証では複数の手順を必要とする可能性があるからです。`authenticate` メソッドでは、呼び出し側は、認証を行い、必要に応じてこのセッションのプリンシパルの属性を選択できます。

セキュリティ・インフラストラクチャが許可していないために呼び出しが失敗すると、標準例外 `CORBA::NO_PERMISSION` が発生します。セキュリティ・インフラストラクチャのインプリメンテーションが機能をサポートしていないためにメソッドが失敗すると、標準例外 `CORBA::NO_IMPLEMENT` が発生します。パラメータに不適切な値を指定すると、標準例外 `CORBA::BAD_PARAM` が発生します。タイミングに関連する問題が起こると、`CORBA::COMM_FAILURE` が発生します。Bootstrap オブジェクトは、ほとんどのシステム例外を `CORBA::Invalid_Domain` にマッピングします。

Principal Authenticator は、位置制約付きオブジェクトです。したがって、CORBA の DII/DSI 機能を介して Principal Authenticator オブジェクトを使用することはできません。このオブジェクトの現在のプロセス外にリファレンスを渡そうとしたり、`CORBA::ORB::object_to_string` を使用してオブジェクトを外部化しようとしたりすると、`CORBA::MARSHAL` 例外が発生します。

証明書による認証での Principal Authenticator オブジェクトの使用

Principal Authenticator オブジェクトは、証明書による認証をサポートするように機能拡張されています。証明書の使用は、`PrincipalAuthenticator::authenticate` オペレーションに対するパラメー

タとして Tobj::CertificateBased の Security::AuthenticationMethod 値を指定することで制御します。証明書による認証を使用する場合、PrincipalAuthenticator::authenticate オペレーションのインプリメンテーションは、プリンシパルの秘密鍵とデジタル証明書を取得し、それを SSL プロトコルで使用するように登録して、プリンシパルのクリデンシャルを取得する必要があります。

PrincipalAuthenticator::authenticate オペレーションの security_name パラメータと auth_data パラメータは、プリンシパルの秘密鍵を開くために使用します。ユーザが、この 2 つのパラメータで適切な値を指定しなかった場合は、秘密鍵を開くことができず、認証は失敗します。秘密鍵が正常に開かれると、プリンシパルのローカル ID を表すデジタル証明書チェーンが構築されます。秘密鍵とデジタル証明書チェーンは、SSL プロトコルで使用されるように登録する必要があります。

BEA TuxedoPrincipal Authenticator オブジェクトに対する拡張

BEA Tuxedo 製品の CORBA 環境は、BEA Tuxedo 製品の ATMI 環境のセキュリティと同様のセキュリティ・メカニズムをサポートするために Principal Authenticator オブジェクトを拡張します。拡張機能は、Tobj::PrincipalAuthenticator インターフェイスを定義することで提供されます。このインターフェイスは、tpinit 関数を介して ATMI 環境で使用できる機能と同様の機能を提供するメソッドを含んでいます。Tobj::PrincipalAuthenticator インターフェイスは、SecurityLevel2::PrincipalAuthenticator インターフェイスから派生したものです。

拡張された Principal Authenticator オブジェクトは、CORBA サービス・セキュリティ・サービス仕様で定義した Principal Authenticator オブジェクトと同じ規則に従います。

拡張された Principal Authenticator オブジェクトのインプリメンテーションでは、ユーザは、認証に使用するユーザ名、クライアント名、追加認証データ (パスワードなど) を提供する必要があります。情報は、ネットワークを介

して IIOP リスナ / ハンドラに送信する必要があるので、信頼性を確保するためにセキュリティ保護されています。保護には、ユーザが提供した暗号化などがあります。

`Tobj::PrincipalAuthenticator` インターフェイスをサポートする拡張された Principal Authenticator オブジェクトは、プリンシパルの認証に `SecurityLevel2::PrincipalAuthenticator` インターフェイスを使用するのと同等の機能を提供します。ただし、`SecurityLevel2::PrincipalAuthenticator::authenticate` メソッドとは異なり、`Tobj::PrincipalAuthenticator` インターフェイスで定義された logon メソッドは、Credentials オブジェクトを返しません。

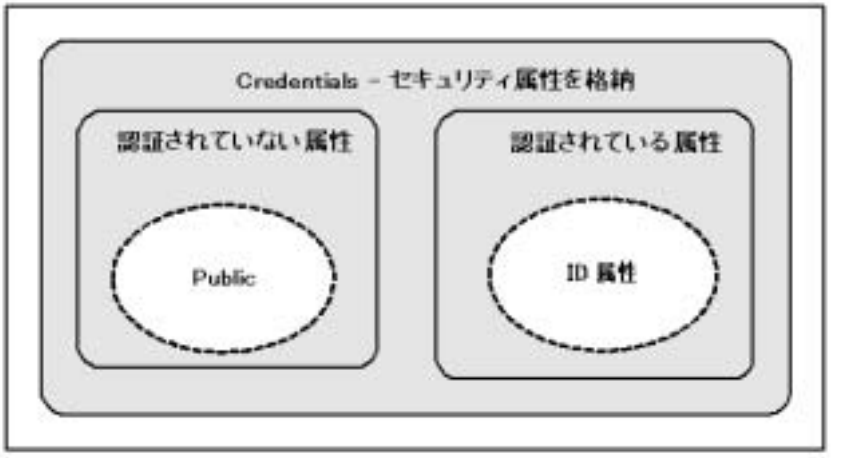
Credentials オブジェクト

Credentials オブジェクト (図 13-2 を参照) は、プリンシパルのセキュリティ属性を保持します。Credentials オブジェクトは、そのオブジェクトが表すプリンシパルのセキュリティ属性を取得および設定するためのメソッドを提供します。これらのセキュリティ属性には、認証済みまたは認証を受けていない ID および権限が含まれています。Credentials オブジェクトにも、セキュリティの関連付けが確立されていない情報が含まれています。

Credentials オブジェクトは、以下のプロセスの結果として作成されます。

- 認証
- 既存の Credentials オブジェクトのコピー
- SecurityCurrent オブジェクトを介した Credentials オブジェクトの要求

図 13-2Credentials オブジェクト



Credentials オブジェクトの複数のリファレンスがサポートされています。Credentials は状態を持つオブジェクトです。Credentials オブジェクトの作成対象となったプリンシパルの代わりに状態を保持します。この情報には、プリンシパルの ID と権限を決定するのに必要な情報が含まれています。Credentials オブジェクトは、オブジェクトを作成した Principal Authenticator オブジェクトに関連付けられていませんが、プリンシパルの ID を認証する認証局を示している必要があります。

Credentials オブジェクトは、位置制約付きオブジェクトです。したがって、DII/DSI 機能を介して Credentials オブジェクトを使用することはできません。このオブジェクトの現在のプロセス外にリファレンスを渡そうとしたり、`CORBA::ORB::object_to_string` を使用してオブジェクトを外部化しようとしたりすると、`CORBA::MARSHAL` 例外が発生します。

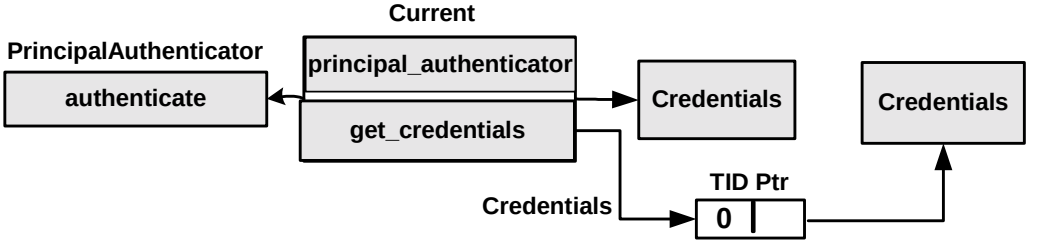
Credentials オブジェクトは、アプリケーション開発者が、セキュリティ保護された接続に対してセキュリティ属性を指定できるように機能拡張されています。これらの属性を使用すると、セキュリティ保護された接続で整合性、信頼性、またはその両方が必要かどうかを指定できます。この機能をサポートするために、`SecurityLevel2::Credentials` インターフェイスに 2 つの属性が追加されています。

- invocation_options_supported 属性は、セキュリティ保護された接続を確立したときに使用可能なセキュリティ・オプションを示します。
- invocation_options_required 属性を使用すると、アプリケーション開発者は、セキュリティ保護された接続の確立に使用する必要がある必要最低限のセキュリティ・オプションを指定できます。

SecurityCurrent オブジェクト

SecurityCurrent オブジェクト (図 13-3 を参照) は、プリンシパルとターゲット・オブジェクトの両方で現在の実行コンテキストを表します。SecurityCurrent オブジェクトは、現在の実行コンテキストに関連付けられたサーバ固有の状態情報を表します。クライアント・アプリケーションとサーバ・アプリケーションの両方が、実行のスレッドに関連付けられた状態と、スレッド実行中のプロセスを表す SecurityCurrent オブジェクトを持っています。

図 13-3SecurityCurrent オブジェクト



SecurityCurrent オブジェクトは、シングルトン・オブジェクトです。つまり、プロセス・アドレス領域で許可されるインスタンスは 1 つだけです。SecurityCurrent オブジェクトの複数のリファレンスがサポートされています。

CORBA サービス・セキュリティ・サービス仕様は、セキュリティに関連付けられている SecurityCurrent オブジェクトのインターフェイスを 2 つ定義します。

- `SecurityLevel1::Current`。`CORBA::Current` から派生したものです。
- `SecurityLevel2::Current`。`SecurityLevel1::Current` から派生したものです。

2 つのインターフェイスは、実行コンテキストに関連付けられたセキュリティ情報へのアクセスを提供します。

どの段階でも、クライアント・アプリケーションは、`Current::get_credentials` メソッドを呼び出して、呼び出しのクリデンシャルを要求することで、以降の呼び出しに対するデフォルト・クリデンシャルを決定できます。デフォルト・クリデンシャルは、オブジェクト・リファレンスを使用するすべての呼び出しで使用されます。

`Current::get_attributes` メソッドがクライアント・アプリケーションによって呼び出された場合、Credentials オブジェクトから返された属性はプリンシパルのものです。

SecurityCurrent オブジェクトは、位置制約付きオブジェクトです。したがって、DII/DSI 機能を介して SecurityCurrent オブジェクトを使用することはできません。このオブジェクトの現在のプロセス外にリファレンスを渡そうとしたり、`CORBA::ORB::object_to_string` を使用してオブジェクトを外部化しようとしたりすると、`CORBA::MARSHAL` 例外が発生します。

章

14 セキュリティ・モジュール

ここでは、CORBA セキュリティ・モデルで使用する以下のモジュールの Object Management Group (OMG) インターフェイス定義言語 (IDL) の定義を説明します。

- CORBA
- TimeBase
- Security
- Security Level 1
- Security Level 2
- Tobj

CORBA モジュール

OMG では、Current 擬似オブジェクトをサポートするために、CORBA::Current インターフェイスを CORBA モジュールに追加しています。この変更によって、CORBA モジュールは、Security Replaceability および Security Level 2 をサポートできるようになりました。

リスト 14-1 では、CORBA::Current インターフェイスの OMG IDL 文を示します。

注記　この情報は、「CORBAservices: Common Object Services Specification」(1995 年 3 月 31 日改訂版、1997 年 11 月更新) の 15-230 ページから、OMG の許可を得て転載しています。

コード リスト 14-1　CORBA::Current **インターフェイスの** OMG IDL **文**

```
module CORBA {
       //CORBA に対する拡張
       interface Current {
       };
};
```

TimeBase モジュール

基本的な Time Service、Universal Time オブジェクト、および Time Interval オブジェクトに適用されるすべてのデータ構造体は、TimeBase モジュールで定義されます。これによって、ほかのサービスは、インターフェイス定義なしでもこのデータ構造を使用できるようになります。インターフェイス定義およびそれに関連する enum と例外は、TimeBase モジュールにカプセル化されます。

リスト 14-2 では、TimeBase モジュールの OMG IDL 文を示します。

注記　この情報は、「CORBAservices: Common Object Services Specification」(1995 年 3 月 31 日改訂版、1997 年 11 月更新) の 14-5 ページから、OMG の許可を得て転載しています。

コード リスト 14-2　TimeBase **モジュールの** OMG IDL **文**

```
//Time Service から
module TimeBase {
       //ulonglong データ型の仮の定義
       // これは、すべてのクライアント ORB による型の拡張の適用を保留
       struct ulonglong {
              unsigned long      low;
              unsigned long      high;
       };
       typedef ulonglong         TimeT;
       typedef short             TdfT;
       struct UtcT {
              TimeT              time;     // 8 オクテット
              unsigned long      inacclo;  // 4 オクテット
              unsigned short     inacchi;  // 2 オクテット
              TdfT               tdf;      // 2 オクテット
                                           // 合計 16 オクテット
       };
};
```

表 14-1 では、TimeBase モジュールのデータ型を定義します。

注記　この情報は、「CORBAservices: Common Object Services Specification」(1995 年 3 月 31 日改訂版、1997 年 11 月更新) の 14-6 ページから、OMG の許可を得て転載しています。

表 14-1 TimeBase モジュールのデータ型定義

データ型	定義
`Time ulonglong`	OMG IDL には現在、符号なし 64 ビット整数を表すネイティブなデータ型はありません。その RFP に対する技術の適用は、符号なし 64 ビット整数を表すネイティブなデータ型を OMG IDL で定義するための手段を提供します。 その技術の適用を保留すると、この構造体を使用して符号なし 64 ビット整数を表し、ネイティブなデータ型が使用可能になったときに、すべてのプラットフォームでこの宣言と相互運用できないことを理解することができます。この定義は一時的なもので、ネイティブな符号なし 64 ビット整数型が OMG IDL で使用できるようになると削除されます。
`Time TimeT`	`TimeT` は、64 ビットの単一の時間値を表し、基準時間から経過した 100 ナノ秒数を保持します。絶対時間では、基準時間は 15 October 1582 00:00 です。
`Time TdfT`	`TdfT` は、16 ビットの short 型で、グリニッジ子午線を基準にした置換の秒数の形式で時間の置換係数を保持します。子午線の東は正数、西は負数で表されます。
`Time UtcT`	`UtcT` は、サービス全般で使用する時間値の構造体です。`UtcT` 構造体が 保持されている場合、相対時間または絶対時間は、履歴によって決まります。その状態情報を保持するオブジェクト内に明示的なフラグはありません。`inacclo` フィールドと `inacchi` フィールドは、48 ビットに圧縮された `InaccuracyT` 型の値を保持します。`tdf` フィールドは、タイム・ゾーン情報を保持します。インプリメンテーションは、Universal Time オブジェクト (UTO) が作成されるときは常に、ローカル・タイム・ゾーンの時間の置換係数をこのフィールドに配置します。 この構造体の内容は、オペークになるように設計されています。正しくマーシャルするために、フィールドのタイプを識別する必要があります。

Security モジュール

Security モジュールは、ほかのセキュリティ・モジュールで共通のセキュリティ・データ型に対する OMG IDL を定義します。このメソッドは、TimeBase モジュールによって異なり、セキュリティ対応の ORB で使用できなければなりません。

リスト 14-3 には、Security モジュールでサポートされているデータ型が示されています。

注記　この情報は、「CORBAservices: Common Object Services Specification」(1995 年 3 月 31 日改訂版、1997 年 11 月更新) の 15-193 ～ 15-195 ページから、OMG の許可を得て転載しています。

コード リスト 14-3　Security **モジュールの** OMG IDL **文**

```
module Security {
        typedef sequence<octet>   Opaque;

       // 標準データ型の拡張可能なファミリ
       struct ExtensibleFamily {
              unsigned short       family_definer;
              unsigned short       family;
        };

       // セキュリティ属性
       typedef unsigned long        SecurityAttributeType;

       //ID 属性 ; ファミリ = 0
       const SecurityAttributeType  AuditId = 1;
       const SecurityAttributeType  AccountingId = 2;
       const SecurityAttributeType  NonRepudiationId = 3;

       // 特権属性 ; ファミリ = 1
       const SecurityAttributeType  Public = 1;
       const SecurityAttributeType  AccessId = 2;
       const SecurityAttributeType  PrimaryGroupId = 3;
       const SecurityAttributeType  GroupId = 4;
       const SecurityAttributeType  Role = 5;
       const SecurityAttributeType  AttributeSet = 6;
       const SecurityAttributeType  Clearance = 7;
       const SecurityAttributeType  Capability = 8;
```

```
        struct AttributeType {
                ExtensibleFamily       attribute_family;
                SecurityAttributeType  attribute_type;
        };

        typedef sequence <AttributeType>   AttributeTypeLists;
        struct SecAttribute {
               AttributeType   attribute_type;
               Opaque          defining_authority;
               Opaque          value;
               // この属性の値は
              // 型の認識を基にしてのみ解釈可能
        };

        typedef sequence<SecAttribute>  AttributeList;

        //Authentication はステータスを返す
        enum AuthenticationStatus {
               SecAuthSuccess,
               SecAuthFailure,
               SecAuthContinue,
               SecAuthExpired
        };

        //Authentication メソッド
         typedef unsigned long   AuthenticationMethod;

         enum CredentialType {
               SecInvocationCredentials;
               SecOwnCredentials;
               SecNRCredentials

        //TimeBase から選択
        typedef TimeBase::UtcT   UtcT;
};
```

表 14-2 では、Security モジュールのデータ型を説明します。

表 14-2 Security モジュールのデータ型定義

データ型	定義
sequence<octet>	表現がセキュリティ・サービス・インプリメンテーションにのみ認識されるデータ。

Security Level 1 モジュール

ここでは、レベル 1 のセキュリティ機能のみを使用するクライアント・アプリケーション・オブジェクトで使用可能なインターフェイスを定義します。このモジュールは、CORBA モジュール、Security モジュール、TimeBase モジュールによって異なります。Current インターフェイスは、ORB によってインプリメントされます。

リスト 14-4 では、Security Level 1 モジュールの OMG IDL 文を示します。

注記　この情報は、「CORBAservices: Common Object Services Specification」(1995 年 3 月 31 日改訂版、1997 年 11 月更新) の 15-198 ページから、OMG の許可を得て転載しています。

コード リスト 14-4　Security Level 1 **モジュールの** OMG IDL **文**

```
module SecurityLevel1 {
       interface Current : CORBA::Current {// PIDL
             Security::AttributeList get_attributes(
               in Security::AttributeTypeList  attributes
            );
       };
};
```

Security Level 2 モジュール

ここでは、レベル 2 のセキュリティ機能を使用するクライアント・アプリケーション・オブジェクトで使用可能な追加インターフェイスを定義します。このモジュールは、CORBA モジュールおよび Security モジュールによって異なります。

リスト 14-5 では、Security Level 2 モジュールの OMG IDL 文を示します。

注記　この情報は、「CORBAservices: Common Object Services Specification」(1995 年 3 月 31 日改訂版、1997 年 11 月更新) の 15-198 ～ 15-200 ページから、OMG の許可を得て転載しています。

コード リスト 14-5　Security Level 2 **モジュールの** OMG IDL **文**

```
module SecurityLevel2 {
       //インターフェイスの宣言を転送
       interface PrincipalAuthenticator;
       interface Credentials;
       interface Current;

      //インターフェイス Principal Authenticator
       interface PrincipalAuthenticator {
            Security::AuthenticationStatus authenticate(
                  in Security::AuthenticationMethod  method,
                  in string                    security_name,
                  in Security::Opaque          auth_data,
                  in Security::AttributeList   privileges,
                  out Credentials              creds,
                  out Security::Opaque         continuation_data,
                  out Security::Opaque         auth_specific_data
             );

             Security::AuthenticationStatus
                        continue_authentication(
                  in Security::Opaque        response_data,
                  inout Credentials          creds,
                  out Security::Opaque       continuation_data,
                  out Security::Opaque       auth_specific_data
             );
        };

       //インターフェイス Credentials
       interface Credentials {
              attribute Security::AssociationOptions
                                invocation_options_supported;
              attribute Security::AssociationOptions
                                 invocation_options_required;
              Security::AttributeList get_attributes(
                   in Security::AttributeTypeList   attributes
              );
              boolean is_valid(
                     out Security::UtcT      expiry_time
              );
```

```
        };

        // インターフェイス Current。SecurityLevel1::Current から派生。
        // このセキュリティ・レベルでの Current の追加オペレーションを提供する。
        // これは ORB によってインプリメントされる
        interface Current : SecurityLevel1::Current { // PIDL
                void set_credentials(
                        in Security::CredentialType   cred_type,
                        in Credentials                cred
                );

                Credentials get_credentials(
                     in Security::CredentialType   cred_type
                );
                readonly attribute PrincipalAuthenticator
                          principal_authenticator;
        };
};
```

Tobj モジュール

ここでは、Tobj モジュールのインターフェイスを定義します。

このモジュールは、ATMI 型の認証をプログラムするのに使用するインターフェイスを提供します。

リスト 14-6 では、Tobj モジュールの OMG IDL 文を示します。

コード リスト 14-6 Tobj **モジュールの** OMG IDL **文**

```
//Tobj 固有の定義

        //get_auth_type () が値を返す
        enum AuthType {
                TOBJ_NOAUTH,
                TOBJ_SYSAUTH,
                TOBJ_APPAUTH
        };

        typedef sequence<octet>    UserAuthData;
```

```
        interface PrincipalAuthenticator :
                SecurityLevel2::PrincipalAuthenticator { // PIDL
                 AuthType get_auth_type();

                Security::AuthenticationStatus logon(
                       in string            user_name,
                       in string            client_name,
                       in string            system_password,
                       in string            user_password,
                       in UserAuthData      user_data
                 );
                 void logoff();

                void build_auth_data(
                       in string                    user_name,
                       in string                    client_name,
                       in string                    system_password,
                       in string                    user_password,
                       in UserAuthData              user_data,
                       out Security::Opaque         auth_data,
                       out Security::AttributeList  privileges
                );
          };
};
```

章

15 C++ セキュリティ・リファレンス

ここでは、CORBA セキュリティの C++ メソッドについて説明します。

SecurityLevel1::Current::get_attributes

概要 Current インターフェイスの属性を返します。

OMG IDL 定義

```
Security::AttributeList get_attributes(
       in Security::AttributeTypeList   attributes
       );
};
```

引数

`attributes`
値が必要とされているセキュリティ属性 (権限属性タイプ) のセット。このリストが空の場合は、すべての属性が返されます。

説明 このメソッドは、プリンシパルのクリデンシャルから特権などの属性を Current インターフェイス用に取得します。

戻り値 次の表は、有効な戻り値について説明しています。

戻り値	説明
`Security::Public`	空 (認証が実行されなかった場合は、Public が返されます)。
`Security::AccessId`	BEA Tuxedo ユーザ名を含む NULL 終了 ASCII 文字列。
`Security::PrimaryGroupId`	BEA Tuxedo プリンシパル名を含む NULL 終了 ASCII 文字列。

注記 `defining_authority` フィールドは常に空です。`UBBCONFIG` ファイルで定義されているセキュリティ・レベルによって、`get_attribute` メソッドの一部の値を使用できない場合があります。追加された 2 つの値 (`Group Id` および `Role`) は、`UBBCONFIG` ファイルでセキュリティ・レベルが `ACL` または `MANDATORY_ACL` に設定されている場合に使用できます。

注記 この情報は、「CORBAservices: Common Object Services Specification」(1995 年 3 月 31 日改訂版、1997 年 11 月更新) の 15-103、104 ページから、OMG の許可を得て転載しています。

SecurityLevel2::PrincipalAuthenticator::authenticate

概要 プリンシパルを認証し、オプションでプリンシパルのクリデンシャルを取得します。

OMG IDL 定義

```
Security::AuthenticationStatus
     authenticate(
       in   Security::AuthenticationMethod  method,
       in   Security::SecurityName          security_name,
       in   Security::Opaque                auth_data,
       in   Security::AttributeList         privileges,
       out  Credentials                     creds,
       out  Security::Opaque                continuation_data,
       out  Security::Opaque                auth_specific_data );
```

引数

`method`

使用するセキュリティ・メカニズム。有効な値は、`Tobj::TuxedoSecurity` および `Tobj::CertificateBased` です。

`security_name`

プリンシパルの ID 情報 (ログオン情報など)。値は、プリンシパルのユーザ名を含む NULL 終了文字列へのポインタである必要があります。文字列は、NULL 文字を除いて 30 文字以内に制限されています。

証明書による認証を使用する場合、この名前は、LDAP を有効にしたディレクトリ・サービスで証明書をルックアップするのに使用します。また、この名前を基に、秘密鍵を格納するファイルの名前が決まります。たとえば、次のように入力します。`milozzi@company.com` は、LDAP を有効にしたディレクトリ・サービスで証明書をルックアップするのに使用する電子メール・アドレス、`milozzi_company.pem` は秘密鍵・ファイル名です。

`auth_data`

パスワードまたは秘密鍵などのプリンシパルの認証。`Tobj:TuxedoSecurity` セキュリティ・メカニズムを指定した場合、この引数の値は、コンフィギュレーションされた認証のレベルによって異なります。`Tobj::CertificateBased` 引数を指定した場合、この引数の値は、プリンシパルの秘密鍵の復号化に使用するパス・フレーズです。

`privileges`

要求された特権属性。

creds
: 新しく作成された Credentials オブジェクトのオブジェクト・リファレンス。オブジェクト・リファレンスは、完全には初期化されません。したがって、オブジェクト・リファレンスは、SecurityLevel2::Current::authenticate メソッドの戻り値が SecAuthSuccess になるまで使用できません。

continuation_data
: SecurityLevel2::Current::authenticate メソッドの戻り値が SecAuthContinue の場合、この引数は、継続する認証の試行情報を格納します。戻り値は常に空です。

auth_specific_data
: 使用する認証サービス固有の情報。戻り値は常に空です。

説明 SecurityLevel2::Current::authenticate メソッドは、プリンシパルを認証し、BEA Tuxedo ドメインでのセッション時にプリンシパルで必要なオプションで特権属性を要求するために、クライアント・アプリケーションによって使用されます。

Tobj::TuxedoSecurity セキュリティ・メカニズムを指定する場合、Tobj::PrincipalAuthenticator::logon オペレーションを呼び出すことで同じ機能を取得できます。このオペレーションは、同じ機能を提供しますが、ATMI 認証セキュリティ・メカニズムで使用するための特別なオペレーションです。

戻り値 次の表は、有効な戻り値について説明しています。

戻り値	説明
SecAuthSuccess	creds 引数が初期化され、使用する準備ができたときに返される新規作成 Credentials オブジェクトのオブジェクト・リファレンス。

戻り値	説明
SecAuthFailure	認証プロセスが矛盾していたか、プロセスでエラーが発生しました。したがって、creds 引数は、Credentials オブジェクトへのオブジェクト・リファレンスを格納しません。 Tobj::TuxedoSecurity セキュリティ・メカニズムを使用した場合、この戻り値は、認証に失敗したか、クライアント・アプリケーションが既に認証されているものの、Tobj::PrincipalAuthenticator::logoff オペレーションまたは Tobj_Bootstrap::destroy_current オペレーションのいずれかを呼び出さなかったことを示します。
SecAuthContinue	認証手順で試行 / 応答メカニズムを使用していることを示します。creds 引数は、一部が初期化された Credentials オブジェクトのオブジェクト・リファレンスを格納します。continuation_data は、試行の詳細を示します。
SecAuthExpired	認証データに、有効性が期限切れの情報が含まれていることを示します。したがって、creds 引数は、Credentials オブジェクトへのオブジェクト・リファレンスを格納しません。 Tobj::TuxedoSecurity セキュリティ・メカニズムを指定した場合、この戻り値は返されません。
CORBA::BAD_PARAM	以下の場合に、CORBA::BAD_PARAM 例外が発生します。 ■ security_name、auth_data、または privileges 引数に値が指定されていない場合。 ■ 入力引数の長さが、引数の最大長を超えている場合。 ■ method 引数の値が Tobj::TuxedoSecurity、auth_data 引数に空の文字列または NULL 文字列として username または clientname が指定されている場合。

SecurityLevel2::Current::set_credentials

概要 クリデンシャルのタイプを設定します。

OMG IDL 定義

```
void set_credentials(
                in Security::CredentialType   cred_type,
                in Credentials                creds
);
```

引数

`cred_type`
設定するクリデンシャルのタイプ (invocation、own、または non-repudiation)。

`creds`
Credentials オブジェクトに対するオブジェクト・リファレンス。これがデフォルトになります。

説明 このメソッドは、`SecInvocationCredentials` にのみ設定できます。それ以外の値に設定すると、`set_credentials` で `CORBA::BAD_PARAM` が発生します。クリデンシャルは、以前の `SecurityLevel2::Current::get_credentials` または `SecurityLevel2::PrincipalAuthenticator::authenticate` の呼び出しで取得済みになっている必要があります。

戻り値 特にありません。

注記 この情報は、「CORBAservices: Common Object Services Specification」(1995 年 3 月 31 日改訂版、1997 年 11 月更新) の 15-104 ページから、OMG の許可を得て転載しています。

SecurityLevel2::Current::get_credentials

概要 クリデンシャルのタイプを取得します。

OMG IDL 定義

```
Credentials get_credentials(
        in Security::CredentialType   cred_type
);
```

引数

`cred_type`
取得するクリデンシャルのタイプ。

説明 この呼び出しは、`SecInvocationCredentials` の取得にのみ使用できます。それ以外の場合に使用すると、`get_credentials` で `CORBA::BAD_PARAM` が発生します。クリデンシャルを使用できない場合、`get_credentials` で `CORBA::BAD_INV_ORDER` が発生します。

戻り値 クライアント・アプリケーションのアクティブなクリデンシャルのみを返します。

注記 この情報は、「CORBAservices: Common Object Services Specification」(1995 年 3 月 31 日改訂版、1997 年 11 月更新) の 15-105 ページから、OMG の許可を得て転載しています。

SecurityLevel2::Current::principal_authenticator

概要 PrincipalAuthenticator を返します。

OMG IDL 定義

```
readonly attribute PrincipalAuthenticator
                       principal_authenticator;
```

説明 principal_authenticator 属性によって返された PrincipalAuthenticator は、実際の型 Tobj::PrincipalAuthenticator によるものです。したがって、Tobj::PrincipalAuthenticator および SecurityLevel2::PrincipalAuthenticator の双方として使用できます。

注記　このメソッドが不正な SecurityCurrent オブジェクトで呼び出された場合、CORBA::BAD_INV_ORDER が発生します。

戻り値 PrincipalAuthenticator を返します。

SecurityLevel2::Credentials

概要 プロセスに特有のプリンシパルのクリデンシャル情報を表します。SecurityLevel2::Credentials インターフェイスをサポートする Credentials オブジェクトは、位置制約付きオブジェクトです。このオブジェクトの位置の外にリファレンスを渡そうとしたり、CORBA::ORB::object_to_string() を使用してオブジェクトを外部化しようとしたりすると、CORBA::Marshall 例外が発生します。

OMG IDL 定義

```
#ifndef _SECURITY_LEVEL_2_IDL
#define _SECURITY_LEVEL_2_IDL

#include <SecurityLevel1.idl>

#pragma prefix “omg.org”

module SecurityLevel2
  {
  interface Credentials
    {
    attribute Security::AssociationOptions
                              invocation_options_supported;
    attribute Security::AssociationOptions
                              invocation_options_required;
Security::AttributeList
    get_attributes(
      in   Security::AttributeTypeList     attributes );

    boolean
    is_valid(
      out  Security::UtcT                  expiry_time );



};
  };
#endif /* _SECURITY_LEVEL_2_IDL */
```

C++ 宣言

```
class SecurityLevel2
  {
  public:
    classCredentials;
    typedefCredentials *Credentials_ptr;

  class  Credentials : public virtual CORBA::Object
    {
```

```
  public:
    static Credentials_ptr _duplicate(Credentials_ptr obj);
    static Credentials_ptr _narrow(CORBA::Object_ptr obj);
    static Credentials_ptr _nil();

    virtual Security::AssociationOptions
       invocation_options_supported() = 0;
    virtual void
       invocation_options_supported(
          const Security::AssociationOptions  options ) = 0;
    virtual Security::AssociationOptions
       invocation_options_required() = 0;
    virtual void
       invocation_options_required(
          const Security::AssociationOptions  options ) = 0;

    virtual Security::AttributeList *
      get_attributes(
        const Security::AttributeTypeList & attributes) = 0;

    virtual CORBA::Boolean
      is_valid( Security::UtcT_out expiry_time) = 0;

  protected:
    Credentials(CORBA::Object_ptr obj = 0);
    virtual ~Credentials() { }

  private:
    Credentials( const Credentials&) { }
    void operator=(const Credentials&) { }
  };  // クラス Credentials
};  // クラス SecurityLevel2
```

SecurityLevel2::Credentials::get_attributes

概要 クリデンシャルに添付された属性リストを取得します。

OMG IDL 定義

```
Security::AttributeList get_attributes(
      in AttributeTypeList   attributes
);
```

引数

`attributes`
値が必要とされているセキュリティ属性 (権限属性タイプ) のセット。このリストが空の場合は、すべての属性が返されます。

説明 このメソッドは、プリンシパルのクリデンシャルに添付された属性リストを返します。属性タイプのリストでは、`AttributeList` に返す属性タイプの値だけを含める必要があります。属性は現在、属性ファミリまたは ID に基づいて返されません。ほとんどの場合、どのインスタンスでもプリンシパルのクリデンシャルの有効なセットは 1 つしかないので、これは、`SecurityLevel1::Current::get_attributes()` を呼び出した場合と同じ結果です。結果は、クリデンシャルが使用中でない場合には異なる可能性があります。

戻り値 属性リストを返します。

注記 この情報は、「CORBAservices: Common Object Services Specification」(1995 年 3 月 31 日改訂版、1997 年 11 月更新) の 15-97 ページから、OMG の許可を得て転載しています。

SecurityLevel2::Credentials::invocation_options_supported

概要 SSL 接続を確立して BEA Tuxedo ドメインでオブジェクトに対する呼び出しを実行する際に使用できるセキュリティ・オプションの最大数を示します。

OMG IDL 定義

```
attribute Security::AssociationOptions
                                    invocation_options_supported;
```

引数 特にありません。

説明 このメソッドは、`SecurityLevel2::Credentials::invocation_options_required` メソッドと組み合わせて使用する必要があります。

以下のセキュリティ・オプションを指定できます。

セキュリティ・オプション	説明
`NoProtection`	SSL プロトコルは、メッセージ保護を提供しません。
`Integrity`	SSL プロトコルは、メッセージの整合性チェックを提供します。メッセージの整合性を保護するのにデジタル署名を使用します。
`Confidentiality`	SSL 接続は、メッセージの機密性を保護します。メッセージの機密性を保護するのに暗号を使用します。
`DetectReplay`	SSL プロトコルはリプレイ検出を提供します。リプレイは、メッセージが検出されずに繰り返し送信されたときに発生します。
`DetectMisordering`	SSL プロトコルは、要求および要求フラグメントのシーケンス・エラーの検出を提供します。
`EstablishTrustInTarget`	要求のターゲットがプリンシパルの開始元に対して自身を認証することを示します。
`NoDelegation`	アクセス制御を決定するために、中間オブジェクトによる特権の使用をプリンシパルが許可することを示します。ただし、プリンシパルの特権は委譲されないので、チェーン内の次のオブジェクトを呼び出すときに中間オブジェクトは特権を使用できません。

セキュリティ・オプション	説明
SimpleDelegation	アクセス制御を決定するために、中間オブジェクトによる特権の使用をプリンシパルが許可することを示し、その中間オブジェクトに特権を委譲します。ターゲット・オブジェクトは、クライアント・アプリケーションの特権のみを受け取り、中間オブジェクトの ID を認識しません。この呼び出しオプションがターゲット・オブジェクトに対して制約なしに使用された場合、動作は偽装として認識されます。
CompositeDelegation	中間オブジェクトによるクリデンシャルの使用および委譲をプリンシパルが許可することを示します。プリンシパルと中間オブジェクトの両方の特権をチェックできます。

戻り値　定義されたセキュリティ・オプションのリスト。

セキュリティの関連付けに Tobj::TuxedoSecurity セキュリティ・メカニズムを使用する場合、NoProtection、EstablishTrustInClient、および SimpleDelegation セキュリティ・オプションのみが返されます。BEA Tuxedo ドメインへのアクセスにパスワードが要求されるように CORBA アプリケーションのセキュリティ・レベルが定義された場合にのみ、EstablishTrustInClient セキュリティ・オプションが表示されます。

注記　指定したセキュリティ・オプションが、CORBA アプリケーション用に定義されたセキュリティ・メカニズムでサポートされていない場合、CORBA::NO_PERMISSION 例外が返されます。この例外は、指定したセキュリティ・オプションが、SecurityLevel2::Credentials::invocation_options_required メソッドで指定したセキュリティ・オプションよりも少ない機能しか持っていない場合にも発生します。

invocation_options_supported 属性には、set() メソッドと get() メソッドがあります。Tobj::TuxedoSecurity セキュリティ・メカニズムを使用して Credentials オブジェクトを取得する場合には、set() メソッドを使用できません。Tobj::TuxedoSecurity セキュリティ・メカニズムで set() メソッドを使用すると、CORBA::NO_PERMISSION 例外が返されます。

SecurityLevel2::Credentials::invocation_options_required

概要 SSL 接続を確立して BEA Tuxedo ドメインでターゲット・オブジェクトに対する呼び出しを実行する際に使用できるセキュリティ・オプションの最少数を指定します。

OMG IDL 定義

```
attribute Security::AssociationOptions
                                    invocation_options_required;
```

引数 特にありません。

説明 このメソッドを使用すると、プリンシパルと BEA Tuxedo ドメインの間の通信の保護を指定できます。このメソッドを使用すると、Credentials オブジェクトは、定義したセキュリティ・オプションのレベルの SSL プロトコルを使用してターゲット・オブジェクトに対する呼び出しを実行します。このメソッドは、`SecurityLevel2::Credentials::invocation_options_supported` メソッドと組み合わせて使用する必要があります。

以下のセキュリティ・オプションを指定できます。

セキュリティ・オプション	説明
`NoProtection`	SSL プロトコルは、メッセージ保護を提供しません。
`Integrity`	SSL プロトコルは、メッセージの整合性チェックを提供します。メッセージの整合性を保護するのにデジタル署名を使用します。
`Confidentiality`	SSL 接続は、メッセージの機密性を保護します。メッセージの機密性を保護するのに暗号を使用します。
`DetectReplay`	SSL プロトコルはリプレイ検出を提供します。リプレイは、メッセージが検出されずに繰り返し送信されたときに発生します。
`DetectMisordering`	SSL プロトコルは、要求および要求フラグメントのシーケンス・エラーの検出を提供します。
`EstablishTrustInTarget`	要求のターゲットがプリンシパルの開始元に対して自身を認証することを示します。

セキュリティ・オプション	説明
NoDelegation	アクセス制御を決定するために、中間オブジェクトによる特権の使用をプリンシパルが許可することを示します。ただし、プリンシパルの特権は委譲されないので、チェーン内の次のオブジェクトを呼び出すときに中間オブジェクトは特権を使用できません。
SimpleDelegation	アクセス制御を決定するために、中間オブジェクトによる特権の使用をプリンシパルが許可することを示し、その中間オブジェクトに特権を委譲します。ターゲット・オブジェクトは、クライアント・アプリケーションの特権のみを受け取り、中間オブジェクトの ID を認識しません。この呼び出しオプションがターゲット・オブジェクトに対して制約なしに使用された場合、動作は偽装として認識されます。
CompositeDelegation	中間オブジェクトによるクリデンシャルの使用および委譲をプリンシパルが許可することを示します。プリンシパルと中間オブジェクトの両方の特権をチェックできます。

戻り値　定義されたセキュリティ・オプションのリスト。

セキュリティの関連付けに Tobj::TuxedoSecurity セキュリティ・メカニズムを使用する場合、NoProtection、EstablishTrustInClient、および SimpleDelegation セキュリティ・オプションのみが返されます。BEA Tuxedo ドメインへのアクセスにパスワードが要求されるように CORBA アプリケーションのセキュリティ・レベルが定義された場合にのみ、EstablishTrustInClient セキュリティ・オプションが表示されます。

注記　指定したセキュリティ・オプションが、CORBA アプリケーション用に定義されたセキュリティ・メカニズムでサポートされていない場合、CORBA::NO_PERMISSION 例外が返されます。この例外は、指定したセキュリティ・オプションが、SecurityLevel2::Credentials::invocation_options_supported メソッドで指定したセキュリティ・オプションよりも多い機能を持っている場合にも発生します。

invocation_options_required 属性には、set() メソッドと get() メソッドがあります。Tobj::TuxedoSecurity セキュリティ・メカニズムを使用して Credentials オブジェクトを取得する場合には、set() メソッドを使用できません。Tobj::TuxedoSecurity セキュリティ・メカニズムで set() メソッドを使用すると、CORBA::NO_PERMISSION 例外が返されます。

SecurityLevel2::Credentials::is_valid

概要 クリデンシャルのステータスをチェックします。

OMG IDL 定義

```
boolean is_valid(
     out Security::UtcT      expiry_time
);
```

説明 使用されたクリデンシャルがそのときにアクティブな場合、このメソッドは、TRUE を返します。つまり、Tobj::PrincipalAuthenticator::logoff または Tobj_Bootstrap::destroy_current を呼び出していないということです。Tobj::PrincipalAuthenticator::logoff() の後にこのメソッドが呼び出された場合、FALSE が返されます。Tobj_Bootstrap::destroy_current() の後にこのメソッドが呼び出された場合、CORBA::BAD_INV_ORDER 例外が発生します。

戻り値 返された有効期限には、maximum unsigned long long 値 (C++) および maximum long (Java) が含まれています。unsigned long long データ型が適用されると、ulonglong が控えのデータ型になります。ulonglong データ型は次のように定義します。

```
//ulonglong データ型の仮の定義
// これは、すべてのクライアント ORB による型の拡張の適用を保留
struct ulonglong {
       unsigned long      low;
       unsigned long      high;
};
```

注記 この情報は、「CORBAservices: Common Object Services Specification」(1995 年 3 月 31 日改訂版、1997 年 11 月更新) の 15-97 ページから、OMG の許可を得て転載しています。

SecurityLevel2::PrincipalAuthenticator

概要 プリンシパルを認証できるようにします。
SecurityLevel2::PrincipalAuthenticator インターフェイスをサポートする Principal Authenticator オブジェクトは、位置制約付きオブジェクトです。このオブジェクトの位置の外にリファレンスを渡そうとしたり、CORBA::ORB::object_to_string() を使用してオブジェクトを外部化しようとしたりすると、CORBA::Marshall 例外が発生します。

OMG IDL 定義

```
#ifndef _SECURITY_LEVEL_2_IDL
#define _SECURITY_LEVEL_2_IDL

#include <SecurityLevel1.idl>

#pragma prefix “omg.org”

module SecurityLevel2
  {
  interface PrincipalAuthenticator
    {    //位置制約付き
    Security::AuthenticationStatus authenticate (
        in   Security::AuthenticationMethod method,
        in   Security::SecurityName         security_name,
        in   Security::Opaque               auth_data,
        in   Security::AttributeList        privileges,
        out  Credentials                    creds,
        out  Security::Opaque               continuation_data,
        out  Security::Opaque               auth_specific_data
    );

    Security::AuthenticationStatus continue_authentication (
        in   Security::Opaque               response_data,
        in   Credentials                    creds,
        out  Security::Opaque               continuation_data,
        out  Security::Opaque               auth_specific_data
    );
    };
  };
#endif // SECURITY_LEVEL_2_IDL


#pragma prefix "beasys.com"
module Tobj
  {
  const Security::AuthenticationMethod
```

```
    TuxedoSecurity = 0x54555800;
    CertificateBased = 0x43455254;
  };
```

C++ 宣言

```
class SecurityLevel2
  {
  public:
    classPrincipalAuthenticator;
    typedefPrincipalAuthenticator * PrincipalAuthenticator_ptr;

  class PrincipalAuthenticator : public virtual CORBA::Object
    {
    public:
      static PrincipalAuthenticator_ptr
        _duplicate(PrincipalAuthenticator_ptr obj);
      static PrincipalAuthenticator_ptr
        _narrow(CORBA::Object_ptr obj);
      static PrincipalAuthenticator_ptr _nil();


      virtual Security::AuthenticationStatus
          authenticate (
            Security::AuthenticationMethod method,
            const char * security_name,
            const Security::Opaque & auth_data,
            const Security::AttributeList & privileges,
            Credentials_out creds,
            Security::Opaque_out continuation_data,
            Security::Opaque_out auth_specific_data) = 0;


      virtual Security::AuthenticationStatus
          continue_authentication (
                     const Security::Opaque & response_data,
                    Credentials_ptr & creds,
                    Security::Opaque_out continuation_data,
                    Security::Opaque_out auth_specific_data) = 0;

    protected:
      PrincipalAuthenticator(CORBA::Object_ptr obj = 0);
      virtual ~PrincipalAuthenticator() { }


    private:
      PrincipalAuthenticator( const PrincipalAuthenticator&) { }
      void operator=(const PrincipalAuthenticator&) { }
    };  // クラス PrincipalAuthenticator
  };
```

SecurityLevel2::PrincipalAuthenticator::continue_authentication

概要 常に失敗します。

OMG IDL 定義

```
Security::AuthenticationStatus continue_authentication(
        in Security::Opaque             response_data,
        in Credentials                  creds,
        out Security::Opaque            continuation_data,
        out Security::Opaque            auth_specific_data
);
```

説明 BEA Tuxedo ソフトウェアは 1 つの手順で認証を実行するため、このメソッドは常に失敗し、`Security::AuthenticationStatus::SecAuthFailure` を返します。

戻り値 常に `Security::AuthenticationStatus::SecAuthFailure` を返します。

注記 この情報は、「CORBAservices: Common Object Services Specification」(1995 年 3 月 31 日改訂版、1997 年 11 月更新) の 15-92、93 ページから、OMG の許可を得て転載しています。

Tobj::PrincipalAuthenticator::get_auth_type

概要 BEA Tuxedo ドメインによって予測されている認証のタイプを取得します。

OMG IDL 定義

```
AuthType get_auth_type();
```

説明 このメソッドは、BEA Tuxedo ドメインによって予測されている認証のタイプを返します。

注記　このメソッドが不正な SecurityCurrent オブジェクトで呼び出された場合、CORBA::BAD_INV_ORDER が発生します。

戻り値 Tobj_AuthType 列挙へのリファレンス。BEA Tuxedo ドメインへのアクセスに必要な認証のタイプを返します。次の表は、有効な戻り値について説明しています。

戻り値	説明
TOBJ_NOAUTH	認証はまったく必要ありません。ただし、クライアント・アプリケーションは、ユーザ名とクライアント・アプリケーション名を指定することで認証を受けることができます。パスワードは不要です。 このセキュリティ・レベルを指定するには、UBBCONFIG ファイルの RESOURCES セクションにある SECURITY パラメータに NONE 値を指定します。
TOBJ_SYSAUTH	クライアント・アプリケーションは、BEA Tuxedo ドメインに対して自身を認証する必要があります。また、ユーザ名、クライアント名、およびパスワードを指定する必要があります。 このセキュリティ・レベルを指定するには、UBBCONFIG ファイルの RESOURCES セクションにある SECURITY パラメータに APP_PW 値を指定します。

戻り値	説明
TOBJ_APPAUTH	クライアント・アプリケーションは、BEA Tuxedo ドメインに対してクライアント・アプリケーションを認証する証明資料を提供する必要があります。証明資料は、デジタル証明書のパスワードの場合もあります。 このセキュリティ・レベルを指定するには、UBBCONFIG ファイルの RESOURCES セクションにある SECURITY パラメータに USER_AUTH 値を指定します。

Tobj::PrincipalAuthenticator::logon

概要 プリンシパルを認証します。

OMG IDL 定義

```
Security::AuthenticationStatus logon(
       in string          user_name,
       in string          client_name,
       in string          system_password,
       in string          user_password,
       in UserAuthData    user_data
);
```

引数

user_name
: BEA Tuxedo ユーザ名です。認証レベルは、TOBJ_NOAUTH です。user_name が NULL または空の場合、あるいは 30 文字を超えている場合、logon で CORBA::BAD_PARAM が発生します。

client_name
: クライアント・アプリケーションの BEA Tuxedo 名です。認証レベルは、TOBJ_NOAUTH です。client_name が NULL または空の場合、あるいは 30 文字を超えている場合、logon で CORBA::BAD_PARAM 例外が発生します。

system_password
: CORBA クライアント・アプリケーション・パスワードです。認証レベルは、TOBJ_SYSAUTH です。クライアント名が NULL または空の場合、あるいは 30 文字を超えている場合、logon で CORBA::BAD_PARAM 例外が発生します。

注記　system_password は、30 文字以内で指定します。

user_password
: デフォルト BEA Tuxedo 認証サービスで使用するのに必要なユーザ・パスワードです。認証レベルは、TOBJ_APPAUTH です。パスワードは 30 文字以内で指定します。

user_data
: カスタム BEA Tuxedo 認証サービスで使用するのに必要なクライアント・アプリケーション固有のデータです。認証レベルは、TOBJ_APPAUTH です。

注記　TOBJ_SYSAUTH には、TOBJ_NOAUTH に加えてクライアント・アプリケーション・パスワードの要件が含まれます。TOBJ_APPAUTH には、TOBJ_SYSAUTH に加えて、ユーザ・パスワードまたはユーザ・データなどの追加情報が含まれます。

注記　user_password 引数と user_data 引数は、相互に排他的であり、BEA Tuxedo ドメインのコンフィギュレーションで使用する認証サービスの要件によって異なります。BEA Tuxedo のデフォルト認証サービスでは、ユーザ・パスワードの方を予期します。カスタマイズされた認証サービスでは、ユーザ・データを要求するものもあります。user_password と user_data の両方が設定されている場合、ログオン呼び出しで CORBA::BAD_PARAM 例外が発生します。

説明　このメソッドは、プリンシパルが BEA Tuxedo ドメインにアクセスできるように IIOP リスナ / ハンドラを介してプリンシパルを認証します。このメソッドは SecurityLevel2::PrincipalAuthenticator::authenticate と同じ機能ですが、引数は ATMI 認証指向です。

注記　このメソッドが不正な SecurityCurrent オブジェクトで呼び出された場合、CORBA::BAD_INV_ORDER が発生します。

戻り値　次の表は、有効な戻り値について説明しています。

戻り値	**説明**
Security::AuthenticationStatus:: SecAuthSuccess	認証が成功したことを示します。

戻り値	**説明**

`Security::AuthenticationStatus::SecAuthFailure`	認証が失敗したか、またはクライアント・アプリケーションが既に認証済みで、以下のメソッドのいずれかを呼び出さなかったことを示します。
	`Tobj::PrincipalAuthenticator:logoff`
	`Tobj_Bootstrap::destroy_current`

Tobj::PrincipalAuthenticator::logoff

概要　プリンシパルに関連付けられたセキュリティ・コンテキストを破棄します。

OMG IDL 定義

```
void logoff();
```

説明　この呼び出しは、セキュリティ・コンテキストを破棄しますが、BEA Tuxedo ドメインへのネットワーク接続を閉じません。`Logoff` でも、現在のクリデンシャルを無効化できます。認証タイプが `TUBJ_NOAUTH` でない場合、ログオフした後の既存のオブジェクト・リファレンスを使用した呼び出しは失敗します。

プリンシパルが現在 BEA Tuxedo ドメインに対して認証されている場合、`Tobj_Bootstrap::destroy_current()` を呼び出すと、暗黙的に `logoff` が呼び出されます。

注記　このメソッドが不正な SecurityCurrent オブジェクトで呼び出された場合、`CORBA::BAD_INV_ORDER` が発生します。

戻り値　特にありません。

Tobj::PrincipalAuthenticator::build_auth_data

概要 SecurityLevel2::PrincipalAuthenticator::authenticate で使用する認証データと属性を作成します。

OMG IDL 定義

```
void build_auth_data(
       in string                    user_name,
       in string                    client_name,
       in string                    system_password,
       in string                    user_password,
       in UserAuthData              user_data,
       out Security::Opaque         auth_data,
       out Security::AttributeList  privileges
);
```

引数

user_name
: BEA Tuxedo ユーザ名です。

client_name
: CORBA クライアント名です。

system_password
: CORBA クライアント・アプリケーション・パスワードです。

user_password
: デフォルト BEA Tuxedo 認証サービスで使用するユーザ・パスワードです。

user_data
: カスタム BEA Tuxedo 認証サービスで使用するクライアント・アプリケーション固有のデータです。

auth_data
: authenticate で使用します。

privileges
: authenticate で使用します。

注記 user_name、client_name、または system_password が NULL または空の場合、あるいは 30 文字を超えている場合、以降の authenticate メソッド呼び出しで CORBA::BAD_PARAM 例外が発生します。

注記　user_password パラメータと user_data パラメータは、相互に排他的であり、BEA Tuxedo ドメインのコンフィギュレーションで使用する認証サービスの要件によって異なります。BEA Tuxedo のデフォルト認証サービスでは、ユーザ・パスワードの方を予期します。カスタマイズされた認証サービスでは、ユーザ・データを要求するものもあります。user_password と user_data の両方が設定されている場合、以降の認証呼び出しで CORBA::BAD_PARAM 例外が発生します。

説明　このメソッドは、SecurityLevel2::PrincipalAuthenticator::authenticate で使用する認証データと属性を作成するヘルパ関数です。

注記　このメソッドが不正な SecurityCurrent オブジェクトで呼び出された場合、CORBA::BAD_INV_ORDER が発生します。

戻り値　特にありません。

章

16 Java セキュリティ・リファレンス

セキュリティ・アプリケーション・プログラミング・インターフェイス (API) の詳細については、BEA Tuxedo オンライン・マニュアルの CORBA Javadoc を参照してください。

章

17 オートメーション・セキュリティ・リファレンス

ここでは、CORBA セキュリティのオートメーション・メソッドについて説明します。また、オートメーション・メソッドを使用して ActiveX クライアント・アプリケーションにセキュリティをインプリメントするプログラミングの例も示します。

ここでは、以下の内容について説明します。

- メソッドの説明
- プログラミングの例

注記　オートメーション・セキュリティ・メソッドは、証明書による認証または SSL プロトコルの使用をサポートしません。

メソッドの説明

ここでは、オートメーション・セキュリティ・サービスのメソッドについて説明します。

DISecurityLevel2_Current

`DISecurityLevel2_Current` オブジェクトは、CORBA セキュリティ・モデルの BEA インプリメンテーションです。BEA Tuxedo ソフトウェアのこのリリースでは、`get_attributes()` メソッド、`set_credentials()` メソッド、`get_credentials()` メソッド、および `Principal_Authenticator()` メソッドがサポートされます。

DISecurityLevel2_Current.get_attributes

概要 Current インターフェイスの属性を返します。

MIDL マッピング

```
HRESULT get_attributes(
  [in] VARIANT attributes,
  [in,out,optional] VARIANT* exceptionInfo,
  [out,retval] VARIANT* returnValue);
```

オートメーション・マッピング

```
Function get_attributes(attributes, [exceptionInfo])
```

パラメータ

`attributes`
: 値が必要とされているセキュリティ属性 (権限属性タイプ) のセット。このリストが空の場合は、すべての属性が返されます。

`exceptioninfo`
: エラー発生時にクライアント・アプリケーションが追加例外データを取得できるようにするオプションの入力引数。ActiveX クライアント・アプリケーションでは、すべての例外データは、OLE オートメーション・エラー・オブジェクトで返されます。

説明 このメソッドは、クライアント・アプリケーションに対するクリデンシャルの特権などの属性を Current インターフェイスから取得します。

戻り値 `DISecurity_SecAttribute` オブジェクトの配列を含むバリアント。次の表は、有効な戻り値について説明しています。

戻り値	説明
`Security::Public`	空 (認証が実行されなかった場合は、Public が返されます)。
`Security::AccessId`	BEA Tuxedo ユーザ名を含む NULL 終了 ASCII 文字列。
`Security::PrimaryGroupId`	BEA Tuxedo クライアント・アプリケーション名を含む NULL 終了 ASCII 文字列。

DISecurityLevel2_Current.set_credentials

概要 クリデンシャルのタイプを設定します。

MIDL マッピング

```
HRESULT set_credentials(
  [in] Security_CredentialType cred_type,
  [in] DISecurityLevel2_Credentials* cred,
  [in,out,optional] VARIANT* exceptionInfo);
```

オートメーション・マッピング

```
Sub set_credentials(cred_type As Security_CredentialType,
                    cred As DISecurityLevel2_Credentials,
                      [exceptionInfo])
```

説明 このメソッドは、SecInvocationCredentials にのみ設定できます。それ以外の値に設定すると、set_credentials で CORBA::BAD_PARAM が発生します。クリデンシャルは、以前の DISecurityLevel2_Current.get_credentials の呼び出しで取得済みになっている必要があります。

引数

cred_type
: 設定するクリデンシャルのタイプ (invocation、own、または non-repudiation)。

cred
: Credentials オブジェクトに対するオブジェクト・リファレンス。これがデフォルトになります。

exceptioninfo
: エラー発生時にクライアント・アプリケーションが追加例外データを取得できるようにするオプションの入力引数。ActiveX クライアント・アプリケーションでは、すべての例外データは、OLE オートメーション・エラー・オブジェクトで返されます。

戻り値 特にありません。

DISecurityLevel2_Current.get_credentials

概要 クリデンシャルのタイプを取得します。

MIDL マッピング

```
HRESULT get_credentials(
  [in] Security_CredentialType cred_type,
  [in,out,optional] VARIANT* exceptionInfo,
  [out,retval] DISecurityLevel2_Credentials** returnValue);
```

オートメーション・マッピング

```
Function get_credentials(cred_type As Security_CredentialType,
                  [exceptionInfo]) As DISecurityLevel2_Credentials
```

説明 この呼び出しは、SecInvocationCredentials の取得にのみ使用できます。それ以外の場合に使用すると、get_credentials で CORBA::BAD_PARAM が発生します。クリデンシャルを使用できない場合、get_credentials で CORBA::BAD_INV_ORDER が発生します。

引数

cred_type
: 取得するクリデンシャルのタイプ。

exceptioninfo
: エラー発生時にクライアント・アプリケーションが追加例外データを取得できるようにするオプションの入力引数。ActiveX クライアント・アプリケーションでは、すべての例外データは、OLE オートメーション・エラー・オブジェクトで返されます。

戻り値 クライアント・アプリケーションのみのアクティブなクリデンシャル用の DISecurityLevel2_Credentials オブジェクト。

DISecurityLevel2_Current.principal_authenticator

概要 `PrincipalAuthenticator` を返します。

MIDL マッピング

```
HRESULT principal_authenticator([out, retval]
            DITobj_PrincipalAuthenticator** returnValue);
```

オートメーション・マッピング

```
Property principal_authenticator As DITobj_PrincipalAuthenticator
```

説明 `principal_authenticator` プロパティによって返された `PrincipalAuthenticator` は、実際の型 `DITobj_PrincipalAuthenticator` によるものです。したがって、これは `DISecurityLevel2_PrincipalAuthenticator` として使用できます。

注記　このメソッドが不正な SecurityCurrent オブジェクトで呼び出された場合、`CORBA::BAD_INV_ORDER` が発生します。

戻り値 `DITobj_PrincipalAuthenticator` オブジェクト。

DITobj_PrincipalAuthenticator

DITobj_PrincipalAuthenticator オブジェクトは、BEA Tuxedo ドメインに対するログインおよびログアウトに使用します。BEA Tuxedo ソフトウェアのこのリリースでは、authenticate メソッド、build_auth_data() メソッド、continue_authentication() メソッド、get_auth_type() メソッド、logon() メソッド、および logoff() メソッドがインプリメントされています。

DITobj_PrincipalAuthenticator.authenticate

概要 クライアント・アプリケーションを認証します。

MIDL マッピング

```
HRESULT authenticate(
     [in] long                                  method,
     [in] BSTR                                  security_name,
     [in] VARIANT                               auth_data,
     [in] VARIANT                               privileges,
     [out] DISecurityLevel2_Credentials**

                                                creds,
     [out] VARIANT*                             continuation_data,
     [out] VARIANT*                             auth_specific_data,
     [in,out,optional] VARIANT*                 exceptionInfo,
     [out,retval] Security_AuthenticationStatus* returnValue);
```

オートメーション・マッピング

```
Function authenticate(method As Long, security_name As String,
      auth_data, privileges, creds As DISecurityLevel2_Credentials,
       continuation_data, auth_specific_data,
       [exceptionInfo]) As Security_AuthenticationStatus
```

引数

`method`
Must be `Tobj::TuxedoSecurity`. If `method` is invalid, `authenticate` raises `CORBA::BAD_PARAM`.

`security_name`
BEA Tuxedo ユーザ名です。

`auth_data`
`DITobj_PrincipalAuthenticator.build_auth_data` によって返されます。`auth_data` が不正の場合、`authenticate` で `CORBA::BAD_PARAM` が発生します。

`privileges`
`DITobj_PrincipalAuthenticator.build_auth_data` によって返されます。`privileges` が不正の場合、`authenticate` で `CORBA::BAD_PARAM` が発生します。

`creds`
SecurityCurrent オブジェクトに配置されます。

`continuation_data`
常に空です。

`auth_specific_data`
常に空です。

`exceptioninfo`
エラー発生時にクライアント・アプリケーションが追加例外データを取得できるようにするオプションの入力引数。ActiveX クライアント・アプリケーションでは、すべての例外データは、OLE オートメーション・エラー・オブジェクトで返されます。

説明 このメソッドは、クライアント・アプリケーションが BEA Tuxedo ドメインにアクセスできるように IIOP リスナ / ハンドラを介してクライアント・アプリケーションを認証します。

戻り値 `Security_AuthenticationStatus` Enum 値。次の表は、有効な戻り値について説明しています。

戻り値	**説明**
`Security::Authentication Status:: SecAuthSuccess`	認証が成功したことを示します。
`Security::Authentication Status:: SecAuthFailure`	認証が失敗したか、またはクライアント・アプリケーションが既に認証済みで、`Tobj::PrincipalAuthenticator:logoff` または `Tobj_Bootstrap::destroy_current` を呼び出さなかったことを示します。

DITobj_PrincipalAuthenticator.build_auth_data

概要 DITobj_PrincipalAuthenticator.authenticate で使用する認証データと属性を作成します。

MIDL マッピング

```
HRESULT build_auth_data(
  [in] BSTR                 user_name,
  [in] BSTR                 client_name,
  [in] BSTR                 system_password,
  [in] BSTR                 user_password,
  [in] VARIANT              user_data,
  [out] VARIANT*            auth_data,
  [out] VARIANT*            privileges,
  [in,out,optional] VARIANT* exceptionInfo);
```

オートメーション・マッピング

```
Sub build_auth_data(user_name As String, client_name As String,
   system_password As String, user_password As String, user_data,
   auth_data, privileges, [exceptionInfo])
```

引数

user_name
: BEA Tuxedo ユーザ名です。

client_name
: CORBA クライアント・アプリケーション名です。

system_password
: CORBA クライアント・アプリケーションのパスワードです。

user_password
: デフォルト認証サービスのユーザ・パスワードです。

user_data
: カスタム認証サービスで使用するクライアント・アプリケーション固有のデータです。

auth_data
: authenticate で使用します。

privileges
: authenticate で使用します。

exceptioninfo
: エラー発生時にクライアント・アプリケーションが追加例外データを取得できるようにするオプションの入力引数。ActiveX クライア

ント・アプリケーションでは、すべての例外データは、OLE オートメーション・エラー・オブジェクトで返されます。

注記　user_name、client_name、または system_password が NULL または空の場合、あるいは 30 文字を超えている場合、以降の authenticate メソッド呼び出しで CORBA::BAD_PARAM 例外が発生します。

注記　user_password パラメータと user_data パラメータは、相互に排他的であり、BEA Tuxedo ドメインのコンフィギュレーションで使用する認証サービスの要件によって異なります。デフォルトの認証サービスでは、ユーザ・パスワードの方を予測します。カスタマイズされた認証サービスでは、ユーザ・データを要求するものもあります。user_password と user_data の両方が設定されている場合、以降の認証呼び出しで CORBA::BAD_PARAM 例外が発生します。

説明　このメソッドは、DITobj_PrincipalAuthenticator.authenticate で使用する認証データと属性を作成するヘルパ関数です。

注記　このメソッドが不正な SecurityCurrent オブジェクトで呼び出された場合、CORBA::BAD_INV_ORDER が発生します。

戻り値　特にありません。

DITobj_PrincipalAuthenticator.continue_authentication

概要 常に Security::AuthenticationStatus::SecAuthFailure を返します。

MIDL マッピング

```
HRESULT continue_authentication(
  [in] VARIANT response_data,
  [in,out] DISecurityLevel2_Credentials** creds,
  [out] VARIANT* continuation_data,
  [out] VARIANT* auth_specific_data,
  [in,out,optional] VARIANT* exceptionInfo,
  [out,retval] Security_AuthenticationStatus* returnValue);
```

オートメーション・マッピング

```
Function continue_authentication(response_data,
   creds As DISecurityLevel2_Credentials, continuation_data,
   auth_specific_data, [exceptionInfo]) As
   Security_AuthenticationStatus
```

説明 BEA Tuxedo ソフトウェアは 1 つの手順で認証を実行するため、このメソッドは常に失敗し、Security::AuthenticationStatus::SecAuthFailure を返します。

戻り値 常に SecAuthFailure を返します。

DITobj_PrincipalAuthenticator.get_auth_type

概要 BEA Tuxedo ドメインによって予測されている認証のタイプを取得します。

MIDL マッピング

```
HRESULT get_auth_type(
        [in, out, optional] VARIANT* exceptionInfo,
        [out, retval] Tobj_AuthType* returnValue);
```

オートメーション・マッピング

```
Function get_auth_type([exceptionInfo]) As Tobj_AuthType
```

引数

exceptioninfo

エラー発生時にクライアント・アプリケーションが追加例外データを取得できるようにするオプションの入力引数。ActiveX クライアント・アプリケーションでは、すべての例外データは、OLE オートメーション・エラー・オブジェクトで返されます。

説明 このメソッドは、BEA Tuxedo ドメインによって予測されている認証のタイプを返します。

注記　このメソッドが不正な SecurityCurrent オブジェクトで呼び出された場合、CORBA::BAD_INV_ORDER が発生します。

戻り値 Tobj_AuthType 列挙へのリファレンス。次の表は、有効な戻り値について説明しています。

戻り値	説明
TOBJ_NOAUTH	認証はまったく必要ありません。ただし、クライアント・アプリケーションは、ユーザ名とクライアント・アプリケーション名を指定することで認証を受けることができます。パスワードは不要です。 このセキュリティ・レベルを指定するには、UBBCONFIG ファイルの RESOURCES セクションにある SECURITY パラメータに NONE 値を指定します。

戻り値	説明
TOBJ_SYSAUTH	クライアント・アプリケーションは、BEA Tuxedo ドメインに対して自身を認証する必要があります。また、ユーザ名、クライアント名、およびパスワードを指定する必要があります。 このセキュリティ・レベルを指定するには、UBBCONFIG ファイルの RESOURCES セクションにある SECURITY パラメータに APP_PW 値を指定します。
TOBJ_APPAUTH	クライアント・アプリケーションは、BEA Tuxedo ドメインに対してクライアント・アプリケーションを認証する証明資料を提供する必要があります。証明資料は、デジタル証明書のパスワードの場合もあります。 このセキュリティ・レベルを指定するには、UBBCONFIG ファイルの RESOURCES セクションにある SECURITY パラメータに USER_AUTH 値を指定します。

DITobj_PrincipalAuthenticator.logon

概要 BEA Tuxedo ドメインにログインします。適切な入力パラメータは、認証レベルによって異なります。

MIDL マッピング

```
HRESULT logon(
  [in] BSTR                                  user_name,
  [in] BSTR                                  client_name,
  [in] BSTR                                  system_password,
  [in] BSTR                                  user_password,
  [in] VARIANT                               user_data,
  [in,out,optional] VARIANT*                 exceptionInfo,
  [out,retval] Security_AuthenticationStatus*
                                             returnValue);
```

オートメーション・マッピング

```
Function logon(user_name As String, client_name As String,
   system_password As String, user_password As String,
   user_data, [exceptionInfo]) As Security_AuthenticationStatus
```

説明 リモート CORBA クライアント・アプリケーションの場合、このメソッドは、リモート・クライアント・アプリケーションが BEA Tuxedo ドメインにアクセスできるように IIOP リスナ / ハンドラを介してクライアント・アプリケーションを認証します。このメソッドは、DITobj_PrincipalAuthenticator.authenticate と同じ機能ですが、パラメータはセキュリティ指向です。

引数

user_name
: BEA Tuxedo ユーザ名です。このパラメータは、TOBJ_NOAUTH、TOBJ_SYSAUTH、および TOBJ_APPAUTH 認証レベルで必須です。

client_name
: CORBA クライアント・アプリケーション名です。このパラメータは、TOBJ_NOAUTH、TOBJ_SYSAUTH、および TOBJ_APPAUTH 認証レベルで必須です。

system_password
: CORBA クライアント・アプリケーションのパスワードです。このパラメータは、TOBJ_SYSAUTH および TOBJ_APPAUTH 認証レベルで必須です。

user_password
デフォルト認証サービスのユーザ・パスワードです。このパラメータは、TOBJ_APPAUTH 認証レベルで必須です。

user_data
: カスタム認証サービスで使用するアプリケーション固有のデータです。このパラメータは、TOBJ_APPAUTH 認証レベルで必須です。

注記　user_name、client_name、または system_password が NULL または空の場合、あるいは 30 文字を超えている場合、以降の authenticate メソッド呼び出しで CORBA::BAD_PARAM 例外が発生します。

注記　認証レベルが TOBJ_APPAUTH の場合、user_password または user_data のいずれか 1 つを提供できます。

exceptioninfo
: エラー発生時にクライアント・アプリケーションが追加例外データを取得できるようにするオプションの入力引数。ActiveX クライアント・アプリケーションでは、すべての例外データは、OLE オートメーション・エラー・オブジェクトで返されます。

戻り値　次の表は、有効な戻り値について説明しています。

戻り値	説明
Security::AuthenticationStatus::SecAuthSuccess	認証が成功したことを示します。
Security::AuthenticationStatus::SecAuthFailure	認証が失敗したか、またはクライアント・アプリケーションが既に認証済みで、以下のメソッドのいずれかを呼び出さなかったことを示します。 Tobj::PrincipalAuthenticator:logoff Tobj_Bootstrap::destroy_current

DITobj_PrincipalAuthenticator.logoff

概要 CORBA クライアント・アプリケーションに関連付けられた現在のセキュリティ・コンテキストを破棄します。

MIDL マッピング

```
HRESULT logoff([in, out, optional] VARIANT* exceptionInfo);
```

オートメーション・マッピング

```
Sub logoff([exceptionInfo])
```

説明 この呼び出しは、CORBA クライアント・アプリケーションに関連付けられたコンテキストを破棄しますが、BEA Tuxedo ドメインへのネットワーク接続をクローズしません。`Logoff` でも、現在のクリデンシャルを無効化できます。認証タイプが `TUBJ_NOAUTH` でない場合、ログオフした後の既存のオブジェクト・リファレンスを使用した呼び出しは失敗します。

クライアント・アプリケーションが現在 BEA Tuxedo ドメインに対して認証されている場合、`Tobj_Bootstrap.destroy_current()` を呼び出すと、暗黙的に `logoff` が呼び出されます。

引数

`exceptioninfo`
エラー発生時にクライアント・アプリケーションが追加例外データを取得できるようにするオプションの入力引数。ActiveX クライアント・アプリケーションでは、すべての例外データは、OLE オートメーション・エラー・オブジェクトで返されます。

戻り値 特にありません。

DISecurityLevel2_Credentials

`DISecurityLevel2_Credentials` オブジェクトは、CORBA セキュリティ・モデルの BEA インプリメンテーションです。BEA Tuxedo ソフトウェアのこのリリースでは、`get_attributes()` メソッドおよび `is_valid()` メソッドがサポートされます。

DISecurityLevel2_Credentials.get_attributes

概要 クリデンシャルに添付された属性リストを取得します。

MIDL マッピング

```
HRESULT get_attributes(
  [in] VARIANT attributes,
  [in,out,optional] VARIANT* exceptionInfo,
  [out,retval] VARIANT* returnValue);
```

オートメーション・マッピング

```
Function get_attributes(attributes, [exceptionInfo])
```

引数

`attributes`
値が必要とされているセキュリティ属性 (権限属性タイプ) のセット。このリストが空の場合は、すべての属性が返されます。

`exceptioninfo`
エラー発生時にクライアント・アプリケーションが追加例外データを取得できるようにするオプションの入力引数。ActiveX クライアント・アプリケーションでは、すべての例外データは、OLE オートメーション・エラー・オブジェクトで返されます。

説明 このメソッドは、クライアント・アプリケーションのクリデンシャルに添付された属性リストを返します。属性タイプのリストでは、`AttributeList` に返す属性タイプの値だけを含める必要があります。属性は現在、属性ファミリまたは ID に基づいて返されません。ほとんどの場合、どのインスタンスでもクライアント・アプリケーションのクリデンシャルの有効なセットは 1 つしかないので、これは、`DISecurityLevel2.Current::get_attributes()` を呼び出した場合と同じ結果です。結果は、クリデンシャルが使用中でない場合には異なる可能性があります。

戻り値 `DISecurity_SecAttribute` オブジェクトの配列を含むバリアント。

DISecurityLevel2_Credentials.is_valid

概要 クリデンシャルのステータスをチェックします。

MIDL マッピング

```
HRESULT is_valid(
  [out] IDispatch** expiry_time,
  [in,out,optional] VARIANT* exceptionInfo,
  [out,retval] VARIANT_BOOL* returnValue
```

オートメーション・マッピング

```
Function is_valid(expiry_time As Object,
   [exceptionInfo]) As Boolean
```

説明 使用されたクリデンシャルがそのときにアクティブな場合、このメソッドは、TRUE を返します。つまり、DITobj_PrincipalAuthenticator.logoff または destroy_current を呼び出していないということです。DITobj_PrincipalAuthenticator.logoff() の後にこのメソッドが呼び出された場合、FALSE が返されます。destroy_current() の後にこのメソッドが呼び出された場合、CORBA::BAD_INV_ORDER 例外が発生します。

戻り値 DITimeBase_UtcT オブジェクトが max に設定された場合に、expiry_time が出力されます。

プログラミングの例

ここでは、以下をインプリメントする ActiveX クライアント・アプリケーションの一部を示します。

- Bootstrap オブジェクトによる SecurityCurrent オブジェクトの取得
- SecurityCurrent オブジェクトからの Principal Authenticator オブジェクトの取得
- Tuxedo 型の認証の使用
- BEA Tuxedo ドメインからのログオフ

コード リスト 17-1　Tuxedo **型認証を使用する** ActiveX **クライアント・アプリケーション**

```
Set objSecurityCurrent = objBootstrap.CreateObject(“Tobj.SecurityCurrent”)
Set objPrincipalAuthenticator = objSecurityCurrent.principal_authenticator

     AuthorityType = objPrincipalAuthenticator.get_auth_type
     If AuthorityType = TOBJ_APPAUTH Then logonStatus =
                                    oPrincipalAuthenticator.Logon(
                                    UserName,_
                                    ClientName,_
                                    SystemPassword,_
                                    UserPassword
                                    User Data)
End If

     objPrincipalAuthenticator.logoff()
```

索引

T

U

あ

い

お

か

き

く

こ

さ

し

そ

ほ

り

ん