セキュリティガイド

Borland VisiBroker® 7.0

Borland®
Excellence Endures™

Borland Software Corporation
20450 Stevens Creek Blvd., Suite 800
Cupertino, CA 95014 USA
www.borland.com

2006 年 5 月 11 日初版発行
著者：Borland Software Corporation
発行：ボーランド株式会社
PDF

目次

第 1 章

Borland VisiBroker の概要

Borland は，CORBA 開発者に向けて，業界最先端の VisiBroker オブジェクトリクエストブローカー（ORB）を活用するために *VisiBroker for Java*，*VisiBroker for C++*，および *VisiBroker for .NET* を提供しています。この 3 つの VisiBroker は CORBA 2.6 仕様の実装です。

VisiBroker の概要

VisiBroker は，CORBA が Java オブジェクトと Java 以外のオブジェクトの間でやり取りする必要がある分散配布で使用されます。幅広いプラットフォーム（ハードウェア，オペレーティングシステム，コンパイラ，および JDK）で使用できます。VisiBroker は，異種環境の分散システムに関連して一般に発生するすべての問題を解決します。

VisiBroker は次のコンポーネントからなります。

- VisiBroker for Java，VisiBroker for C++，および VisiBroker for .NET（業界最先端のオブジェクトリクエストブローカーの 3 つの実装）。
- VisiNaming Service - Interoperable Naming Specification バージョン 1.3 の完全な実装。
- GateKeeper - ファイアウォールの背後の CORBA サーバーとの接続を管理するプロキシサーバー。
- VisiBroker Console - CORBA 環境を簡単に管理できる GUI ツール。
- コモンオブジェクトサービス - VisiNotify（通知サービス仕様の実装），VisiTransact（トランザクションサービス仕様の実装），VisiTelcoLog（Telecom ログサービス仕様の実装），VisiTime（タイムサービス仕様の実装），VisiSecure など。

VisiBroker の機能

VisiBroker には次の機能があります。

- セキュリティと Web 接続性を容易に装備できます。
- J2EE プラットフォームにシームレスに統合できます（CORBA クライアントが EJB に直接アクセスできる）。
- 堅牢なネーミングサービス（VisiNaming）とキャッシュ，永続的ストレージ，および複製によって高可用性を実現します。
- プライマリサーバーにアクセスできない場合に，クライアントをバックアップサーバーに自動的にフェイルオーバーします。
- CORBA サーバークラスタ内で負荷分散を行います。
- OMG CORBA 2.6 仕様に完全に準拠します。
- Borland JBuilder 統合開発環境と統合されます。
- Borland AppServer などの他の Borland 製品と最適に統合されます。

VisiBroker のマニュアル

VisiBroker のマニュアルセットは次のマニュアルで構成されています。

- *Borland VisiBroker インストールガイド* — VisiBroker をネットワークにインストールする方法について説明します。このマニュアルは，Windows または UNIX オペレーティングシステムに精通しているシステム管理者を対象としています。
- *Borland VisiBroker セキュリティガイド* — VisiSecure for VisiBroker for Java および VisiBroker for C++ など，VisiBroker のセキュリティを確保するための Borland のフレームワークについて説明しています。
- *Borland VisiBroker for Java 開発者ガイド* — Java による VisiBroker アプリケーションの開発方法について記載されています。Visibroker ORB の設定と管理，およびプログラミングツールの使用方法について説明します。また, IDL コンパイラ, スマートエージェント，ロケーションサービス，ネーミングサービス，イベントサービス，オブジェクトアクティベーションデーモン (OAD), Quality of Service (QoS), インターフェースリポジトリ，および Web サービスサポートについても説明します。
- *Borland VisiBroker for C++ 開発者ガイド* — C++ による VisiBroker アプリケーションの開発方法について記載されています。Visibroker ORB の設定と管理，およびプログラミングツールの使用方法について説明します。また, IDL コンパイラ, スマートエージェント，ロケーションサービス，ネーミングサービス，イベントサービス，OAD, QoS，プラグイン可能トランスポートインターフェース，RT CORBA 拡張機能，Web サービスサポート，およびインターフェースリポジトリについても説明します。
- *Borland VisiBroker for .NET 開発者ガイド* — .NET 環境による VisiBroker アプリケーションの開発方法について記載されています。
- *Borland VisiBroker for C++ API リファレンス* — VisiBroker for C++ に付属するクラスとインターフェースについて説明します。
- *Borland VisiBroker VisiTime ガイド* — Borland による OMG Time Service 仕様の実装について説明します。
- *Borland VisiBroker VisiNotify ガイド* — Borland による OMG 通知サービス仕様の実装について説明します。通知メッセージフレームワークの主な機能として, 特に Quality of Service (QoS) のプロパティ, フィルタリング, および Publish/Subscribe Adapter (PSA) の使用方法が記載されています。

- *Borland VisiBroker VisiTransact ガイド* — Borland による OMG Object Transaction Service 仕様の実装および Borland Integrated Transaction Service コンポーネントについて説明します。
- *Borland VisiBroker VisiTelcoLog ガイド* — Borland による OMG Telecom Log Service 仕様の実装について説明します。
- *Borland VisiBroker GateKeeper ガイド* — Web ブラウザやファイアウォールによるセキュリティ制約の下で，VisiBroker GateKeeper を使用して，VisiBroker のクライアントがネットワークを介してサーバーとの通信を確立する方法について説明します。

通常，マニュアルにアクセスするには，VisiBroker とともにインストールされるヘルプビューアを使用します。ヘルプは，スタンドアロンのヘルプビューアからアクセスすることも，VisiBroker コンソールからアクセスすることもできます。どちらの場合も，ヘルプビューアを起動すると独立したウィンドウが表示されるため，このウィンドウからヘルプビューアのメインツールバーにアクセスしてナビゲーションや印刷を行ったり，ナビゲーションペインにアクセスすることができます。ヘルプビューアのナビゲーションペインには，すべての VisiBroker ブックとリファレンス文書の目次，完全なインデックス，および包括的な検索を実行できるページがあります。

重要 Web サイト http://www.borland.com/techpubs には，PDF 版のマニュアルと最新の製品マニュアルがあります。

スタンドアロンヘルプビューアからの VisiBroker オンラインヘルプトピックへのアクセス

製品がインストールされているコンピュータでスタンドアロンのヘルプビューアからオンラインヘルプにアクセスするには，次のいずれかの手順を実行します。

Windows

- ［スタート｜プログラム｜ Borland VisiBroker ｜ Help Topics］の順に選択します。
- または，コマンドプロンプトを開き，製品のインストールディレクトリの ¥bin ディレクトリに移動し，次のコマンドを入力します。

```
help
```

UNIX コマンドシェルを開き，製品のインストールディレクトリの /bin ディレクトリに移動し，次のコマンドを入力します。

```
help
```

ヒント UNIX システムにインストールするときの指定で，PATH エントリのデフォルトに bin を含まないようにします。カスタムインストールオプションを選択して PATH エントリのデフォルトを変更せず，PATH に現在のディレクトリのエントリがない場合は，./help を使用してヘルプビューアを起動できます。

VisiBroker コンソールからの VisiBroker オンラインヘルプトピックへのアクセス

VisiBroker コンソールから VisiBroker オンラインヘルプトピックにアクセスするには，［Help ｜ Help Topics］を選択します。

［Help］メニューには，オンラインヘルプ内のいくつかの文書へのショートカットもあります。ショートカットの 1 つを選択すると，ヘルプトピックビューアが起動し，［Help］メニューで選択した項目が表示されます。

マニュアルの表記規則

VisiBroker のマニュアルでは，文中の特定の部分を表すために，次の表に示す書体と記号を使用します。

表 1.1　マニュアルの表記規則

表記規則	用途
italic	新規の用語およびマニュアル名に使用されます。
`computer`	ユーザーやアプリケーションが提供する情報，サンプルコマンドライン，およびコードです。
bold computer	本文では，ユーザーが入力する情報を示します。サンプルコードでは，重要なステートメントを強調表示します。
[ ]	省略可能な項目。
...	繰り返しが可能な直前の引数。
|	二者択一の選択。

プラットフォームの表記

VisiBroker マニュアルでは，次の記号を使用してプラットフォーム固有の情報を示します。

表 1.2　プラットフォームの表記

記号	意味
Windows	サポートされているすべての Windows プラットフォーム
Win2003	Windows 2003 のみ
WinXP	Windows XP のみ
Win2000	Windows 2000 のみ
UNIX	すべての UNIX プラットフォーム
Solaris	Solaris のみ
Linux	Linux のみ

Borland サポートへの連絡

ボーランド社は各種のサポートオプションを用意しています。それらにはインターネット上の無償サービスが含まれており，大規模な情報ベースを検索したり，他の Borland 製品ユーザーからの情報を得ることができます。さらに Borland 製品のインストールに関するサポートから有償のコンサルタントレベルのサポートおよび高レベルなアシスタンスに至るまでの複数のカテゴリから，電話サポートの種類を選択できます。

Borland のサポートサービスの詳細や Borland テクニカルサポートへの問い合わせについては，Web サイト `http://support.borland.com` で地域を選択してください。

ボーランド社のサポートへの連絡にあたっては，次の情報を用意してください。

- 名前
- 会社名およびサイト ID
- 電話番号
- ユーザー ID 番号（米国のみ）
- オペレーティングシステムおよびバージョン
- Borland 製品名およびバージョン
- 適用済みのパッチまたはサービスパック
- クライアントの言語とそのバージョン（使用している場合）
- データベースとそのバージョン（使用している場合）

- 発生した問題の詳細な内容と経緯
- 問題を示すログファイル
- 発生したエラーメッセージまたは例外の詳細な内容

オンラインリソース

ネットワーク上の次のサイトから情報を得ることができます。

Web サイト http://www.borland.com/jp/

オンラインサポート http://support.borland.com（ユーザー ID が必要）

リストサーバー 電子ニュースレター（英文）を購読する場合は，次のサイトに用意されているオンライン書式を使用してください。

http://www.borland.com/products/newsletters

Web サイト

定期的に http://www.borland.com/jp/products/visibroker/index.html をチェックしてください。VisiBroker 製品チームによるホワイトペーパー，競合製品の分析，FAQ の回答，サンプルアプリケーション，最新ソフトウェア，最新のマニュアル，および新旧製品に関する情報が掲載されます。

特に，次の URL をチェックすることをお勧めします。

- http://www.borland.com/products/downloads/download_visibroker.html（最新の VisiBroker ソフトウェアおよび他のファイル）
- http://www.borland.com/techpubs（マニュアルの更新および PDF）
- http://info.borland.com/devsupport/bdp/faq/（VisiBroker の FAQ）
- http://community.borland.com（英語，開発者向けの弊社 Web ベースニュースマガジン）

Borland ニュースグループ

Borland VisiBroker を対象とした数多くのニュースグループに参加できます。VisiBroker などの Borland 製品のユーザーによるニュースグループへの参加については，http://www.borland.com/newsgroups を参照してください。

メモ これらのニュースグループはユーザーによって管理されているものであり，ボーランド社の公式サイトではありません。

第 2 章

セキュリティの概要

今日では，より多くの企業がインターネットを使った分散アプリケーションで事業を運営しているため，クオリティが高いアプリケーションセキュリティに対するニーズがさらに高まっています。

クレジットカードの番号や預金の残高などの機密情報は，インターネット接続を利用して Web ブラウザと商用 Web サーバーとの間を日常的にやり取りされます。たとえば，インターネットを利用して銀行と取引するユーザーにとって，次の点が確実でなければなりません。

- 銀行を装った違法サイトではなく，自分の口座がある銀行のサーバーと実際に通信をしていること。
- 銀行と交換するデータが，ネットワークに忍び込んで不正に情報を得ようとする不正ユーザーにはわからない形式になっていること。
- 銀行のソフトウェアとやり取りするデータが，不正な修正が加えられたりせずに元の状態のまま届くこと。たとえば，50,000 円の手形の支払い指図が誤ってあるいは不正に 5,000,000 円に変えられることがあってはなりません。

VisiSecure では，クライアントは銀行のサーバーを認証できます。銀行側のサーバーもセキュリティで保護された接続を利用してクライアントを認証できます。従来のアプリケーションでは，接続が確立されると，クライアントは認証のためにユーザー名とパスワードを送信します。この方法は現在でも使用され，VisiSecure の接続が確立されると，ユーザー名とパスワードの交換を暗号化して安全性を高めています。VisiSecure は任意の数の認証領域をサポートしているので，分散アプリケーションの各部にアクセスできます。さらに，VisiSecure を使用すると，アプリケーションのアクセスコントロールルールを表す承認ドメインを作成できます。

VisiSecure の概要

VisiSecure は，VisiBroker および BDOC にセキュリティ保護のフレームワークを提供します。VisiSecure を使用すると，クライアントとサーバーの間にセキュリティで保護された接続を確立できます。

VisiSecure for Java

VisiSecure は 100% Java で作成され，J2EE 1.3 仕様のすべてのセキュリティ要件をサポートします。VisiSecure は認証に JAAS（Java Authentication and Authorization System），SSL 通信に JSSE（Java Secure Socket Extension），および暗号化操作に JCE（Java Cryptograpy Extension）を使用します。JAVA アプリケーション用の大半の API には，既存の JDK または新しく提供された Java 標準 API が取り入れられています。別々のセキュリティ層で，API が重複しないように注意する必要があります。VisiSecure 機能セットは，J2EE 1.3 のセキュリティ要件を超える場合もあります。

VisiSecure for C++

VisiSecure for C++ は，VisiSecure for Java と同様の機能を提供します。詳細については，第 11 章「VisiSecure for C++ API」および第 10 章「セキュリティプロパティ（C++）」を参照してください。

接続性

VisiSecure では，多くのセキュリティテクノロジをプラグインできます。接続性は，さまざまなレベルで提供されます。セキュリティサービスプロバイダは，セキュリティサービスセット全体をプラグインして置き換えることができ，アプリケーション開発者は，より小さなモジュールをプラグインして各自の環境をカスタマイズして統合できます。プラグインを使用できない唯一の層は，VisiBroker ORB の内部インプリメンテーションに綿密に統合され，相互に緊密に対話する CSIv2 層とトランスポート層です。

VisiSecure 設計の柔軟性

VisiSecure は，現在および今後のさまざまなアーキテクチャをサポートできるように，各種のアプリケーションアーキテクチャで動作するように設計されています。ただし，VisiSecure は強力なセキュリティアーキテクチャを提供しますが，サーバーを単独で完全に保護することはできません。物理的なセキュリティを確保し，基本となる Web サーバー（ホスト）およびオペレーティングシステムサービスをできるだけ安全な状態に設定することは，各自の責任です。

VisiSecure for Java の機能

VisiSecure には，次のような機能があります。

- **Enterprise JavaBeans（EJB）コンテナの統合**：VisiSecure は，CORBA セキュリティサービスと CSIv2 を基盤にして，EJB セキュリティメカニズムをシームレスに統合できます。CORBA は，Bean のセキュリティアーキテクチャの機能を強化します。VisiSecure を利用することで，比較的簡単な EJB セキュリティモデルに追加オプションを設定できます。
- **Web コンテナの統合**：VisiSecure は Web コンテナにメカニズムを提供して Web コンテナと統合することにより，必要に応じて認証エンジンと承認エンジンがセキュリティ情報をほかの EJB コンテナへ送信できるようにしています。たとえば，EJB コンテナの Bean を呼び出そうとしているサーブレットは，最初の要求を送信した元のブラウザクライアントのかわりに動作します。クライアントが提供するセキュリティ情報は，シームレスに EJB コンテナへ伝達されます。さらに，Web コンテナの認証と認証エンジンは，Borland の提供する第 3 章「認証」と第 4 章「承認」を使用できるように設定できます。
- **セキュリティサービスアドミニストレータ**：VisiSecure の管理と設定は，使いやすいプロパティを使って実行し，Java Keytool などのツールをサポートします。

- **GateKeeper** : GateKeeper は高レベルのファイアウォールを越えて，認証を伴う接続を可能にします。サーバーとアプリケーションクライアントがファイアウォールを挟んで設置されている場合でも，クライアントはサーバーに接続することができます。GateKeeper の使い方については，『VisiBroker GateKeeper ガイド』を参照してください。
- **Secure Transport Layer** : VisiSecure は，セキュリティで保護されたトランスポート層として SSL を使用します。SSL は，セキュリティで保護された主要な転送レベルのインターネットの通信プロトコルです。SSL では，信頼モデル利用した，メッセージの機密性，メッセージの完全性，および証明書ベースの認証を使用できます。

VisiSecure for C++ の機能

VisiSecure for C++ には，次のような機能があります。

- **認証と承認** : 認証と承認のモデルは，VisiSecure for Java と同様です。これにより，VisiSecure for C++ アプリケーションの機能が拡張されます。
- **セキュリティサービスアドミニストレータ** : VisiSecure の管理と設定は，使いやすい第 10 章「セキュリティプロパティ（C++）」を使って実行します。
- **Secure Transport Layer** : VisiSecure は，セキュリティで保護されたトランスポート層として SSL を使用します。SSL は，セキュリティで保護された主要な転送レベルのインターネットの通信プロトコルです。SSL では，信頼モデル利用した，メッセージの機密性，メッセージの完全性，および証明書ベースの認証を使用できます。

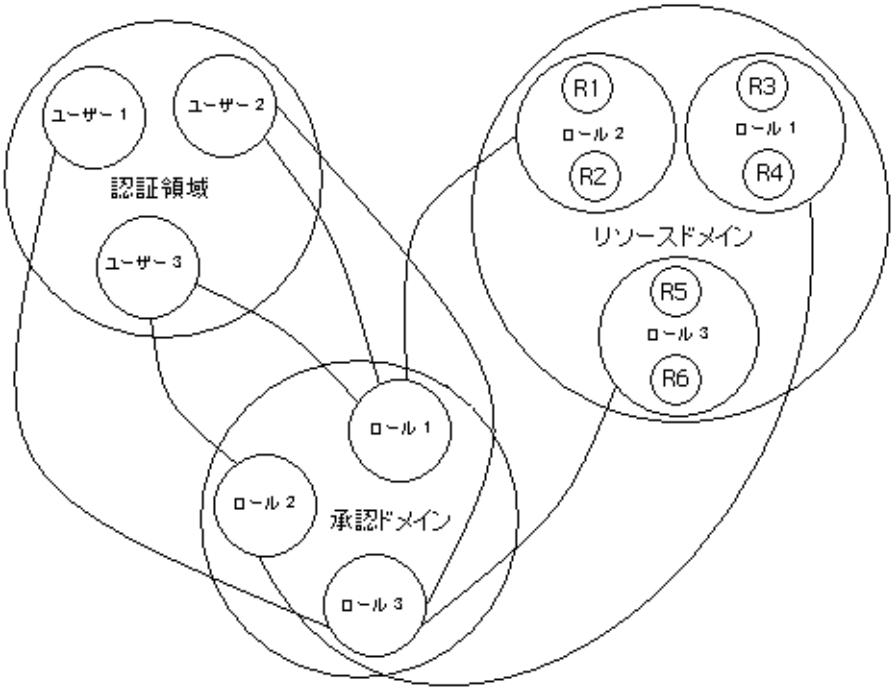
基本セキュリティモデル

基本セキュリティモデルは，ユーザーの視点から見た VisiSecure およびそのコンポーネントを表します。これは，VisiSecure ユーザーが理解し，設定し，対話する必要がある論理モデルです。セキュリティサービスによって，システムのエンティティは次の 3 つの論理グループ（ドメイン）に分類されます。

- 認証領域（ユーザードメイン）：単純なユーザーのデータベースです。各認証領域は，ユーザーのグループ，およびそれに対応する認証情報と特権の属性を表します。
- リソースドメイン：1 つのアプリケーションに関するリソースの集合を表します。アプリケーション開発者は，アプリケーションのリソースに対するアクセスコントロールポリシーを定義します。
- 承認ドメイン：特定のリソースへのアクセスを許可するかどうかを決定する一連のルールを定義します。

次の図に，上記のドメインの関係を示します。

図 2.1 VisiSecure におけるドメイン間の対話



上の図の 3 つの VisiSecure ドメインは密接に関連しています。

1. 認証に対しては認証領域が必要です。VisiBroker には簡単な認証領域が付属します。また，LDAP サーバーなどのサポートされている既存の領域を使用することもできます。
2. 承認に対しては，ロールをセットアップし，ロールにユーザーを関連付ける必要があります。
3. 次に，リソースドメインをセットアップし，そのドメインのリソースへのアクセスを特定のロールに許可する必要があります。

認証領域（ユーザードメイン）

簡単に説明すると，認証領域はユーザーのデータベースです。各認証領域には，ユーザーの集合およびそれに対応する認証情報と権限が記述されます（ユーザーのパスワード，ユーザーが所属するグループなど）。認証領域の例としては，NT ドメイン，NIS，yp データベース，LDAP サーバーなどがあります。

認証領域は，使用する認証テクノロジ，およびデータソースをポイントする一連の設定オプションによって定義されます。たとえば，LDAP を使用する認証領域は，認証プロトコルとして LDAP を指定し，サーバーの名前を指定し，その他の設定パラメータを指定します。システムは，ログオン時にユーザーを認証します。詳細については，12 ページの「認証と識別」を参照してください。

リソースドメイン

*リソース*には，VisiSecure がセキュリティで保護する必要があるアプリケーションコンポーネントを定義します。VisiSecure は，アプリケーションのすべてのリソースが格納されている*リソースドメイン*にリソースを整理します。つまり，サーバーによって公開されるすべてのリモートメソッドやサーブレットは本質的にリソースです。

アプリケーション開発者は，アプリケーションのリソースに対するアクセスコントロールポリシーを定義します。これらは，ロールとして定義されます。*ロール*は，一連のリソースにアクセスするためのアクセス許可の論理的な集合を提供します。詳細については，第 4 章「承認」を参照してください。

さらに，フィールドなどのより詳細なリソースまたはデータベースなどの外部リソースに対するアクセスコントロールを提供して，セキュリティを強化するアプリケーションもあります。EJB とサーブレットの仕様は標準の配布デスクリプタ情報を提供するので，アプリケーションはこれを使用して，各自のアクセスポリシーを特定のメソッドにアクセスするために必要なロールの集合として定義できます。

承認ドメイン

承認ドメインでは，ユーザーは与えられたロールで作業できます。VisiSecure は，ロールに基づいてリソースにアクセスするための権限を許可します。VisiBroker アプリケーションがアプリケーション間でユーザー ID を渡す場合，ID にはユーザー情報および指定されたロールに基づいたアクセス許可が入っています。呼び出し元の ID は要求されているルールと照合され，要求されているルールを満たすかどうかが判定されます。呼び出し元がルールを満たすと，アクセスが許可されます。それ以外の場合，アクセスは拒否されます。詳細については，18 ページの「承認」を参照してください。

分散された環境と VisiSecure SPI

分散環境では，基本セキュリティモデルを構成する 3 つのドメインに加えて，次の項目を考慮する必要があります。

- 認証と識別
- ID アサーション

VisiSecure SPI（Service Provider Interface）は，セキュリティで保護された転送，アサーション，およびアサーションの信頼を処理するためのインターフェースとクラスを提供します。転送（または相互運用性）は，基底の CSIv2 インプリメンテーションによって処理されます。SPI のインプリメンテーションは VisiBroker ORB に密接にバンドルされているので，ほかの言語のための汎用 SPI としてコアから分離することはできません。

特に，VisiSecure SPI のクラスを使用すると，次のセキュリティサービスをカスタマイズできます。

- 識別と認証
- 承認（またはアクセスコントロールの意志決定）
- アサーションの信頼

JAAS による認証と承認の管理

Java 認証承認サービス（JAAS）は，プラグイン可能な*承認*とユーザーベースの*認証*を可能にする拡張機能を定義します。このフレームワークによって承認と認証のインプリメンテーションを効果的に分離できるので，柔軟性とベンダーサポートを強化できます。細粒度のアクセスコントロール機能によって，アプリケーション開発者は適切な粒度レベルで重要なリソースへのアクセスを制御できます。

認証と識別

認証は，エンティティ（ユーザー，サービス，コンポーネントなど）の身元が正しいかどうかを確認するプロセスです。認証プロセスには，次の内容が含まれます。

1 認証を求めているエンティティからの認証情報の取得

2 認証情報の検証

VisiSecure は JAAS フレームワークを使ってエンティティとシステム間の対話を実現します。

システムの識別

すべてのシステムは，リソースへのアクセスが許可される前に身元を提示する必要があります。リソースへのアクセスには，常にクライアントの識別が必要です。CORBA/J2EE 環境では，サーバーの身元の確認も必ず必要になります。サーバーは，次の 2 つの場合に身元の確認が必要です。

- 1. TSL（Transport Layer Security）に SSL を使用する場合，一般にサーバーはクライアントに身元を提示する必要があります。
- 2. 中間層サーバーがほかの中間層サーバーまたは最終層サーバーを呼び出す場合，呼び出し元のかわりに動作する前に身元を提示する必要があります。

詳細については，第 3 章「認証」を参照してください。

認証と接続性

VisiBroker における認証は，プラグインによる認証を可能にする JAAS のインプリメンテーションです。JAAS ログオンサービスは，インプリメンテーションと設定を分離します。第 3 章「認証」という低レベルのシステムプログラミングインターフェースは，プラグイン可能セキュリティモジュールのためのアンカーポイントを提供します。

同時に，システムの認証として，セキュリティサブシステムの各種のコンポーネント間の認証情報の通信（または転送）のための「形式」を表すために認証メカニズムの概念が使用されます。認証／識別プロセスのセキュリティサービスプロバイダは，基底のコアシステムが使用する形式（プロセスのエンコードとデコード）を実装します。

分散環境では，エンティティおよびそれに対応する認証情報の表現を一般的な方法を使ってピア間で転送する必要があるので，認証プロセスはさらに複雑になります。そのために，VisiSecure Java SPI は `AuthenticationMechanism` の概念を使用し，分散環境で認証／識別を行うための一連のクラスを定義します。

サーバーおよびクライアントの認証

JAAS の VisiBroker のインプリメンテーションでは，さまざまな認証メカニズムを設定できます。サーバー認証を使用する一方で，14 ページの「証明書と証明機関」を使ってクライアントからサーバーが認証を受けるようにすることもできます。また，クライアント認証も使用できます。クライアントは，パスワードまたは公開キー証明書を使って認証されます。つまり，サーバーは，パスワードを持ったクライアント，または公開キー証明書を持ったクライアントを認証するように設定することができます。

ユーザー名とパスワードによるクライアントの認証

サーバー側の認証が不要な場合には，標準ユーザー名とパスワードの組み合わせを使って認証を行います。ユーザー名とパスワードを使ってクライアントを認証するには，いくつかの条件があります。サーバーは，クライアントを認証できる一連の領域を公開する必要があります。各領域は，実際に認証を行う JAAS LoginModule に対応させる必要があります。最後に，クライアントは，認証を受けるためにユーザー名，パスワード，および領域を提示する必要があります。詳細については，第 3 章「認証」を参照してください。

認証プロパティの設定

認証ポリシー - サーバー認証とクライアント認証のどちらにするか，また公開キー証明書とパスワードのどちらを使って認証を行うかをプロパティの設定によって決めます。詳細については，第 10 章「セキュリティプロパティ（C++）」と第 9 章「セキュリティプロパティ（Java）」を参照してください。

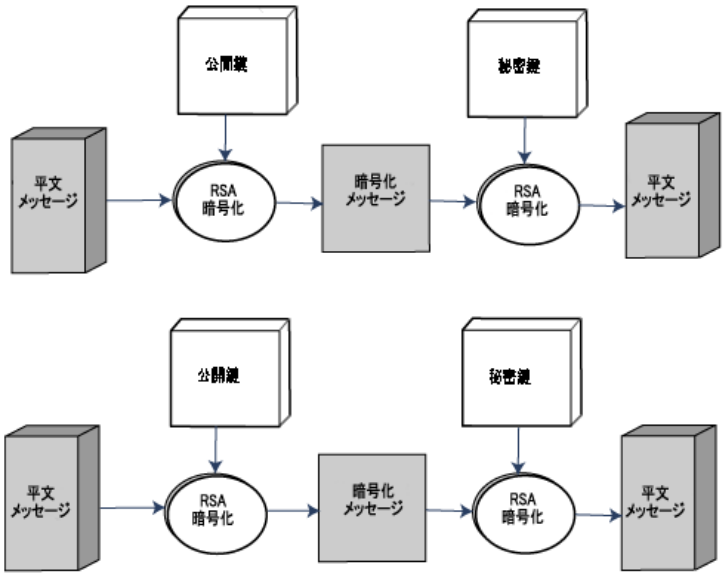
公開キー暗号

VisiSecure は，ユーザー名とパスワードベースの認証のほかに*公開キーの暗号化*もサポートします。公開キーの暗号化では，各ユーザーは*公開*キーと*秘密*キーの 2 つのキーを保有します。公開キーは一般に公開しますが，秘密キーは秘密のままにしておきます。

一般に，暗号化されていないデータを**平文**，暗号化されているデータを**暗号文**と呼びます。公開キーと秘密キーを公開キー暗号化アルゴリズムとともに使用して，次の図のように互いに逆の操作を行います。

- まず，公開キーを使って平文メッセージを暗号文メッセージに暗号化します。一方，秘密キーを使って暗号文メッセージの暗号を解除します。
- 次に，秘密キーを使ってメッセージを暗号化し（一般的な例としては，デジタル署名されたメッセージ），公開キーを使用してこのメッセージの暗号を解除します。

図 2.2　公開キー暗号化アルゴリズムによる公開キーと秘密キーの使用



極秘データを送るには送り先の公開キーを取得し，このキーを使ってデータを暗号化します。暗号化されたデータは，秘密キーを使用しなければ解除できません。
たとえデータの送り主であっても，データの暗号は解除できません。暗号化は*非対称*にも*対称*にもできます。

非対称暗号化

非対称暗号化には公開キーと秘密キーの両方を使用します。両方のキーはともにリンクしているため，公開キーでは暗号化だけができ秘密キーでは暗号解除だけができます。したがって公開キーでは暗号解除はできず，秘密キーでは暗号化はできません。非対称暗号化は，セキュリティ保護に最も優れた暗号化形式です。

対称暗号化

対称暗号化では，1 つのキーを暗号化と暗号解除の両方に使用します。対称暗号化は非対称暗号化より高速ですが，キーを交換するために事前に安全なチャネルが必要で，単一の通信だけが可能です。

証明書と証明機関

公開キーを配布する場合，キーの受信側は送信元のユーザーがだれであるかを確認する必要があります。*ISO X.509 標準*は*証明書*と呼ばれるメカニズムを定義します。この証明書には，*証明機関*（CA）と呼ばれる信頼された機関によってデジタル署名されたユーザーの公開キーがあります。クライアントアプリケーションがサーバーから証明書を受信するか，あるいはこの逆の場合，証明書を発行した CA を使用すると，その証明書を発行したことを確認することができます。CA は公証人のような役目を果たし，証明書は認証を受けた文書のようなものです。

証明書は，15 ページの「秘密キーと証明書要求の生成」して CA に送信して取得します。

デジタル署名

デジタル署名は，目的は基本的に手書きの署名とよく似ています。デジタル署名は一意の署名者を識別するために行います。デジタル署名はさまざまなメソッドを介して作成できます。現在，よく実行されるメソッドの 1 つにデータの暗号化ハッシュがあります。

1. 送信側は，送信用の一方向ハッシュデータを提供します。
2. 送信側は，秘密キーでハッシュを暗号化することで，データにデジタル署名します。
3. 送信側は暗号化ハッシュとオリジナルデータを受信側に送信します。
4. 受信側は，送信側の公開キーを使って暗号化ハッシュを暗号解除します。
5. 受信側は，送信側と同じハッシュのアルゴリズムを使用して，一方向のハッシュデータを提供します。
6. オリジナルハッシュと派生ハッシュが同じであればこのデジタル署名は有効で，文書は変更されずに署名され，公開キーの所有者によって作成されたことが証明できます。

秘密キーと証明書要求の生成

アプリケーションで使用する証明書を入手するには，まず秘密キーと証明書要求を生成する必要があります。このプロセスを自動化するには，Java アプリケーションに対しては Java Keytool を使用し，C++ アプリケーションに対しては OpenSSL ユーティリティなどのオープンソースツールを使用できます。

ファイルを生成した後，証明機関（CA）に対して証明書要求を送信しなければなりません。証明書要求を CA へ提出する手順は，使用している証明書認証によって異なります。社内の CA を使用する場合は，システム管理者に問い合わせてください。外部の商業用 CA を利用する場合は，直接その CA に連絡して証明書要求の送信について指示を受けてください。CA へ送信する証明書要求には，公開キーと識別名が必要です。

識別名

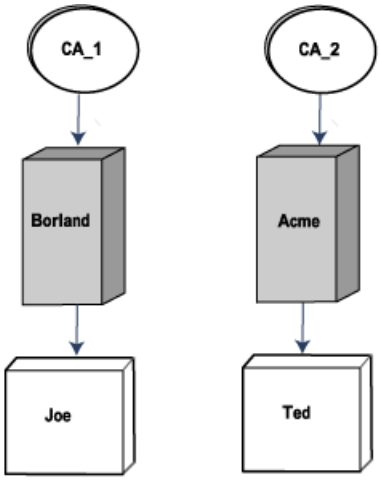
*識別名*はユーザー名，またはユーザーの証明書を発行した CA の名称です。送信する証明書要求には，次の表の要素から構成されたユーザーの識別名が入っています。

タグ	説明	必須要素
Common-Name	このユーザーに関連付ける名前。	はい
Organization	ユーザーの企業または組織名。	はい
Country	ユーザーの居住地を識別する 2 文字の国コード。	はい
Email	ユーザーに関する詳細情報の問い合わせ先。	いいえ
Phone	ユーザーの電話番号。	いいえ
Organizational Unit	ユーザーの所属する部署。	いいえ
Locality	ユーザーの居住している市町村名。	いいえ

証明書チェイン

ISO X.509 標準は，通信しようとしているピアどうしの証明書が別々 CA から発行されている場合のしくみが規定されています。たとえば，次の図では，Joe と Ted は別々の CA の発行した証明書を持っています。

図 2.3　異なる証明機関から発行された証明書



Ted の証明書の有効性を確認しようとする Joe は，信頼されている CA が見つかるまでチェインにある CA をそれぞれ調べる必要があります。信頼されている CA が見つからないと，接続を受け入れるか拒否するかはサーバーが選択します。上の図の Joe の場合は，次のようになります。

1. Joe は Ted の証明書を取得し，発行元 CA が Acme であることを知ります。
2. Joe の証明書チェインには Acme CA がないため，Joe は CA_2 の証明書の発行元を確認します。
3. CA_2 は信頼された CA ではないため，サーバーがその接続を受け入れるか拒否するかどうかを決定します。

メモ　CA からの証明書情報を取得する方法は，その CA に定義されています。

証明書認証

認証と密接に関連しているのが，信頼という概念です。運用面では，信頼は認証と同じように実行されます。証明書の ID が提示されると，信頼はトランスポートレベルで適用されるか，またはさらに高いレベル（CSIv2 層）で適用され，ID はユーザー名とパスワードの形式になります。

Java　Java コードを使って証明書に対して信頼を行うために，VisiSecure は証明書チェインが信頼されているかどうかを示すためにユーザーが用意する JSSE X509TrustManager を使用できるようにするメカニズムを備えています。標準的な Java プロパティを使用して，証明書エントリが信頼されている Java キーストアを指定することもできます。

C++　VisiBroker for C++ ユーザー向けとして，Trustpoints（信頼されている証明書）の設定が可能な API も付属しています。詳細については，第 11 章「VisiSecure for C++ API」を参照してください。

証明書失効リスト（CRL：Certificate Revocation List）と失効した証明書シリアル番号

C++ のみ　署名された公開キー証明書が証明機関によって作成されると，各証明書には失効する期日を示す有効期限が設定されます。ただし，証明書が有効期限前に何らかの理由で失効する場合に対応するために，VisiSecure for C++ には CRL（証明書失効リスト）機能があります。CA（証明機関）の詳細については，14 ページの「証明書と証明機関」を参照してください。

VisiSecure for C++ の CRL 機能を使用すると，CRL を設定し，SSL ハンドシェーク通信時にピアの証明書をこのリストと照合できます。

CRL ファイルは DER 形式で，1 つのディレクトリに保存されている必要があります。CRL を VisiSecure for C++ に追加するには，vbroker.security.CRLRepository プロパティを CRL ファイルがあるディレクトリに設定する必要があります。VisiSecure for C++ のプロパティの詳細については，第 10 章「セキュリティプロパティ（C++）」を参照してください。

メモ　CRL リポジトリディレクトリ構造には複数の CRL ファイルを置くことができます。

CRL がロードされると，VisiSecure はハンドシェーク時にピアが送信するすべての証明書を調べます。ピアの証明書が CRL にある場合は，例外が生成されて接続が拒絶されます。

保護品質（QoP）パラメータのネゴシエーション

クライアントとサーバーの通信では，提供される保護品質（QoP）のいくつかのパラメータにおいて同意する必要があります。リソースホスト（サーバー）は次の処理を実行します。

- サポートできるすべての QoP パラメータを発行します。
- 一連の必須 QoP パラメータをクライアントに要求します。

メモ　定義上，必須 QoP はサポートされている QoP でもあります。

たとえば，サーバーはセキュリティで保護された転送（SSL）をサポートして要求しながら，認証はサポートしても要求しない場合があります。これは，一部のリソースが機密情報ではなく，匿名アクセスを許可できる場合などに便利です。QoP と QoS パラメータの詳細については，次を参照してください。

C++　QoP API

Java　「com.borland.security.csiv2 パッケージ」と第 9 章「セキュリティプロパティ（Java）」

セキュリティで保護された転送

2 つの転送環境における VisiSecure 機能

- 通常のソケットでの IIOP の使用
- セキュアソケット（SSL）の使用

イントラネットでは，通常のソケットで IIOP を使ってユーザー認証情報などの機密データを含めた情報を転送してもセキュリティ上問題がない場合があります。インターネットなどの場合，またはイントラネットの場合であっても，ネットワーク環境が信頼されていない場合は，ネットワークを通して転送されるメッセージの整合性（転送中にメッセージが変更または改竄されていないこと）および秘密性（転送中にメッセージが傍受された場合も読み取ることができないこと）を保証する必要があります。これは，セキュアソケット（SSL）を使って実現します。

JSSE と SSL の接続性

Java　VisiSecure は，SSL 通信を実行するために JSSE（Java Secure Sockets Extension）を使用します。VisiSecure SPI Secure Socket Provider のクラスは，基底の SSL インプリメンテーションへのアクセスを提供します。JSSE（Java Secure Socket Extension）フレームワークに準拠する任意のインプリメンテーションは，適切であれば，ほかのプロバイダメカニズムとは独立して簡単にプラグインできます。対応する JSSE インプリメンテーションに定義されたインターフェース（またはコールバックメソッド）をマッピングする手順だけは必要です。SPI Secure Socket Provider クラスの詳細については，「VisiSecure SPI for Java」と第 12 章「Security SPI for C++」を参照してください。

VisiBroker の「初期」インストールには，Java SDK が提供する JSSE インプリメンテーションを使用します。

暗号化レベルの設定

SSL 製品では多くの暗号化メカニズムを使用しています。このようなメカニズムは，認証，機密，メッセージの整合性アルゴリズムを業界基準に合わせて組み合わせたものです。このような組み合わせを*暗号化スイート*と呼びます。

クライアントとサーバーは，サポートしている暗号化スイートの静的リストを保持しています。ハンドシェークフェーズでこのリストを使用して，どの暗号化スイートを採用するかを決定します。クライアントは，認識できる暗号化スイートのリストをサーバーに送ります。情報を受け取ったサーバーは，サーバーとクライアントの両方で理解できる暗号化スイートを決定します。サーバーは，デフォルトで最も強力な暗号化スイート（セキュリティによる保護レベルが高い暗号化スイート）を選択します。

この暗号化スイート順によって強力なセキュリティが保証されますが，アプリケーション固有のセキュリティ要件に基づいて，ほかの暗号化スイート順を採用することもできます。暗号化スイートの順序を変更するには，16 ページの「保護品質（QoP）パラメータのネゴシエーション」API 関数呼び出しを使用します。これを使用して，現在使用可能な暗号化スイートのリストを取得し，新しい順序に合わせて一覧を設定します。これで，強力な暗号化スイートの前にまず弱い暗号化スイートを使用するようになります。

メモ　新しい暗号スイートを追加することはできません。使用できる暗号スイートの優先順位の変更と，使用しない暗号スイートの削除はできます。

サポートされている暗号化スイート

*暗号化スイート*は，データを暗号化するために使用する有効なエンコードアルゴリズムの集合です。暗号化スイートでは，目的に合わせてさまざまなセキュリティレベルを使用できます。たとえば，ある暗号では認証を提供し，ほかの暗号は提供しません。あるものは暗号化する機能を提供しますが，ほかのものは提供しません。暗号名のセグメントは，暗号化スイートが提供する（または提供しない）内容を示します。

次の表は，暗号名のセグメントとその意味を示しています。

暗号名	説明
RC4（RC8 利用）	暗号で使用する対称的暗号化。
MD5	データ整合性のメカニズム。
	データは平文で送信されますが，データ整合性を確認するために受信の最後でハッシュコードを使用します。
SHA	データ整合性のメカニズム。
WITH	暗号化を伴う認証。
ANON	匿名キー交換アルゴリズムの DLT を使用します。
NULL	暗号化なし。
EXPORT	公開キーのサイズには制限があります。
	メモ：公開キーと秘密キーのサイズが大きくなるほど，キーの安全性も高くなります。このオプションは，一般に米国以外の国のユーザーに対して使用されます。
EXPORT1024	キーの最大サイズは 1024 バイトです。

VisiSecure for Java に対してサポートされている暗号は，使用している JSSE パッケージによって決まります。VisiSecure for C++ のためのリストは，製品に付属する `csstring.h` ファイルにあります。

承認

承認は，ユーザーが身元を提示してから行われます（認証後）。承認は，認証されたエンティティが要求したリソースに対するアクセスコントロールを特定のセキュリティ属性または権限に基づいて決定するプロセスです。VisiBroker は，Java Security Architecture にしたがって承認に*アクセス許可*の概念を使用します。VisiSecure では，リソース承認の判断はアクセス許可に基づきます。Borland は，ユーザーとロールに基づいた専用の承認フレームワークを使って承認を行います。たとえば，クライアントが CORBA または Web 要求エンタープライズ Bean メソッドにアクセスする場合，アプリケーションサーバーはクライアントユーザーがアクセスする権限を持っているかどうかを確認する必要があります。このプロセスをアクセスコントロールあるいは承認と呼びます。

アクセスコントロールリスト

承認はユーザー ID およびロールのリストである*アクセスコントロールリスト*（ACL）に基づいて行われます。通常，アクセスコントロールリストには特定のリソースを使用できる一連のロールを指定します。また，属性が特定のロールと一致するユーザーを指定することにより，このユーザーがロールを実行することができます。

ロールベースのアクセスコントロール

VisiSecure は，ロールに基づいてアクセスコントロール方式を使用します。配布デスクリプタには，各エンタープライズ Bean メソッドへのアクセスの承認を受けているロールのリストが格納されています。VisiSecure では，ロールデータベース（ロール DB）というファイルを使用します。このファイルは，デフォルト名が `roles.db` で，ユーザー ID と EJB のロールとを対応付けます。最低 1 つのロールに関連付けられているユーザーは，メソッドにアクセスすることができます。詳細については，第 4 章「承認」を参照してください。

プラグイン可能な承認

VisiSecure では，ユーザーをロールにマッピングできる承認サービスをプラグインできます。この承認サービスの実装元は，特定のリソースへのアクセスを許可されたオブジェクトの集合を提供します。「ロール」をアクセス許可として表すために，`RolePermission` という新しいクラスが定義されます。次に，承認サービスプロバイダは，指定された権限と特定のリソースの間の関連付けのための `RolePermissions` の同種コレクションへのインプリメンテーションを提供します。

承認サービスは，承認ドメインの概念に密接に対応しています。つまり，各ドメインに実装される承認サービスプロバイダは 1 つです。承認ドメインは，VisiSecure システムと承認サービスインプリメンテーションのブリッジです。ORB（Object Request Broker）自身の初期化中に `vbroker.security.authDomains` プロパティによって定義された承認ドメインが構築され，その間に承認サービスプロバイダインプリメンテーションのインスタンスが作成されます。

承認ドメインは，一連の規則を定義して，ユーザーが論理「ロール」に属すかどうかを判別します。

詳細については，第 4 章「承認」を参照してください。

コンテキスト伝達

転送されたメッセージの秘密性と整合性を保証することに加えて，呼び出し元の ID と認証の情報をクライアントとサーバー間で通信する必要があります。これを「*デリゲーション*」と呼びます。呼び出しパスに複数の層がある場合は，呼び出し元の ID を保守することも必要です。これは，中間層システムへの 1 つの呼び出しがほかのシステムでの呼び出しにつながることがあり，それらは最初の呼び出し元に認可された権限に基づいて実行される必要があるためです。

分散環境では，一般に中間層サーバーが *ID アサーション*を実行して呼び出し元の代理として動作します。最終層サーバーは，アサーションが信頼できるかどうかを判定する必要があります。コンテキストの伝達では，クライアントは次の情報を転送します。

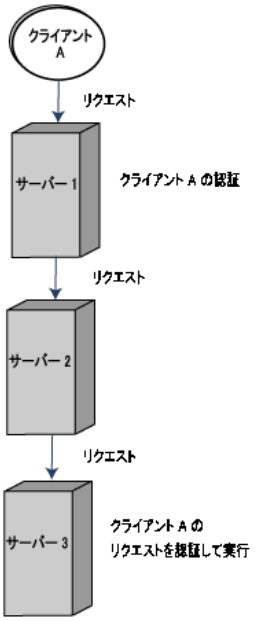
- 認証と識別**トークン** - クライアントの ID と認証情報。
- **ID トークン** - このクライアントが作成した *ID アサーション*。
- **承認要素** - クライアントが呼び出し元または自分自身に関してプッシュする権限情報。

ID アサーション

*ID アサーション*は，通常，セキュリティで保護されたコンポーネントを含む複数のサーバーでクライアントの要求を実行する場合に発生します。サーバー間でクライアントの要求をやり取りする場合など，サーバーがクライアントのかわりに動作しなければならない場合があります。典型的な例としては，クライアントが中間層サーバーを呼び出し，そのサーバーがさらに最終層サーバーを呼び出してクライアントが要求したサービスの一部を実行する場合があります。このような場合，中間層サーバーは一般にクライアントのかわりに動作する必要があります。つまり，中間層サーバーと最終層サーバーの通信におけるアクセスコントロールの決定は，最初の呼び出し元の権限に基づいて行う必要があることを最終層サーバーに知らせる必要があります。

たとえば，クライアントの要求は Server 1 へ送信され，Server 1 はクライアント ID の認証を実行します。ところが，Server 1 は Server 3 へ要求を渡す可能性のある Server 2 へその要求を渡すこともあります。詳細については，次の図を参照してください。

図 2.4　ID アサーション



その後のサーバー（Server 2 と Server 3）は，クライアント ID が Server 1 によって検証されたので，その ID が信頼できるものであることを確認できます。Server 3 のように，最終的にクライアントの要求を実行するサーバーは，18 ページの「承認」だけを実行する必要があります。

デフォルトでは，ID は最初の層のサーバーだけで認証されてアサートされます。これは，ほかの層へ伝達される，アサーションされた ID です。

偽装

*偽装*は，中間層サーバーが最終層サーバーの任意のリソースにアクセスできる ID アサーションの形式です。中間層サーバーは，クライアントのかわりに任意のタスクを実行できます。

デリゲーション

*デリゲーション*は偽装とは異なり，クライアントが明示的に特定の権限をサーバーに委任する ID アサーションの形式です。この場合，サーバーはクライアントが指定した特定のアクションだけを実行できます。VisiSecure は，簡単なデリゲーションだけを実行します。

アサーションの信頼

サーバー（最終層）は，20 ページの「ID アサーション」の受け入れまたは拒否を選択できます。ID アサーションを受け入れる場合は，自身の身元を提示する信頼の問題があります。ピアが正統であることがわかっている場合も，サーバーはピアが別の呼び出し元をアサートできるか，または呼び出し元のかわりに動作するための権限を持っていることを確認する必要があります。最終層は呼び出し元自体を認証せずに中間層のアサーションを受け入れるので，最終層は中間層が最初の呼び出し元に対して適切な 12 ページの「認証と識別」を実行したことを信頼する必要があります。これにより，最終層は中間層の呼び出し元の身元保証を信頼します。

最終層システムには多くのピアが存在する場合があり，一部は中間層として信頼され，一部は単なるクライアントです。したがって，ほかの呼び出し元と対話する権限は，特定のピアだけに許可する必要があります。

アサーションの信頼とプラグイン

リモートピア（サーバーまたはプロセス）が呼び出し元のかわりに動作する際に ID アサーションを行う場合，最終層サーバーがこのアサーションを行うにはピアを信頼する必要があります。SPI（Security Provider Interface）では，Trust Service Provider をプラグインし，指定された呼び出し元および与えられた一連の特権によって，アサーションができる（信頼されている）かどうかを判定できます。つまり，`TrustProvider` クラスを使って信頼ルールを実装し，サーバーが指定されたアサート元サブジェクトから ID アサーションを受け入れるかどうかを判定します。詳細については，「VisiSecure for Java SPI」と第 12 章「Security SPI for C++」を参照してください。

後方信頼

*後方信頼*は初期状態で提供され，何を信頼してアサーションを実行するかを決定するルールをサーバーが保持する形式の信頼です。後方信頼では，クライアントは中間層サーバーが代理で動作する特権を持っているかどうかを知ることはできません。

前方信頼

*前方信頼*は，クライアントが特定の中間層サーバーに明示的に特権を提供するという点でデリゲーションに似ています。

一時特権

サーバーは権限のあるリソースにアクセスして，クライアントのかわりにサービスを実行することがあります。ただし，クライアント自身はリソースにアクセスする権限を持っていない場合があります。一般に，呼び出しにおけるすべてのリソースへのアクセスは，最初の呼び出し元の ID に基づいて評価されます。したがって，この場合は，最初の呼び出し元がリソースに対する権限を持っていないのでアクセスできません。このような状況をサポートするために，アプリケーションは，一時的にサービスを実行している間，呼び出し元とは異なる ID で実行されることがあります。通常，この ID は*論理ロール*と呼ばれます。このユーザーが所属しなければならないロールにより，アプリケーションは，実際に必要なすべてのリソースにアクセスすることができます。

IIOP/HTTPS の使い方

多くのブラウザでサポートしている機能の HTTPS を使用できます。使用するには，次のガイドラインにしたがいます。

- VisiBroker プロキシサーバーの GateKeeper は，外部対応の SSL とともに実行する必要があります。
- IIOP/HTTPS だけを使用するアプレットは，ブラウザやビューアが HTTPS 対応の場合に限って，クライアントにソフトウェア（クラス，またはネイティブライブラリのどちらか）がまだインストールされていない状態でなければなりません。
- IIOP/HTTPS を使用するアプレットは，ID を設定するために `X509Certificate[]` クラスを使用することはできません。すべての証明書と秘密キーの管理はブラウザによって処理されます。さらに，アプレットタグ内の `ORBalwaysTunnel` パラメータが `true` に設定されると，ORB は `SSLCurrent` オブジェクトを解決できません。
- アプレットが IIOP/HTTPS だけを使用するようにするには，HTML ページで `ORBalwaysTunnel` を `true` に設定します。`ORBalwaysTunnel` を `false` に設定するか，または何も設定しない場合，ORB はまず IIOP/SSL を使用しようとします。それには，ローカルにインストールされた SSL クラスとネイティブ SSL ライブラリが必要です。
- 一般に，HTTPS は JDK でサポートされていないので，IIOP/HTTPS は Java アプリケーションに対して有効ではありません。ただし，JDK で HTTPS が使用できなかったり，Java アプリケーションで IIOP/HTTPS が利用できなかったりするような制限は，VisiBroker for Java にはありません。

Netscape Communicator ／ Navigator

IIOP/HTTPS と Netscape Communicator を自由に使用することができますが，Navigator のバージョンによっては IIOP/HTTPS 接続を許可する前に，CA 証明書のインストールを必要とすることがあります。Netscape Navigator で IIOP/HTTPS を使用する場合は，次のガイドラインにしたがいます。

- サーバーの証明書が，Navigator がすでに信頼している CA の発行したものであることを確認してください。
- 信頼された証明書として，Navigator にルート証明書をインストールしてください。証明書ファイル（たとえば cacert.crt の bank_https など）を開くと，証明書をインストールすることができます。
- ブラウザへルート証明書をダウンロードするには GateKeeper を使用してください。その方法については，次に説明します。
- 商用 CA は通常，ルート証明書をインストールためのリンクを用意しています。
- デフォルトでは GateKeeper は，クライアント ID を確認しません。GateKeeper 設定ファイルで，ssl_request_client_certificate を true に設定すると，この確認機能が有効になります。

Microsoft Internet Explorer

Microsoft Internet Explorer で IIOP/HTTPS を使用するには，HTTPS 接続でユーザーと対話していないことを確認してください。たとえば，信頼されていないルート証明書を持っている HTTPS サイトにブラウザがアクセスする場合，HTTPS 接続を確立する前にアクセスの確認を求めてきます。Microsoft JVM の明らかな不具合のために，この接続は失敗します。

接続に失敗する状況とその解決策は，次のとおりです。

- Internet Explorer には信頼できるネットワークサーバー証明機関のリストが付属しています。サーバーの証明書が信頼できる CA（たとえば，bank_https とともに出荷された証明書）から発行されていない場合，HTTPS 接続を確立する前に Internet Explorer から接続の確認を求められます。Microsoft JVM はユーザーによる操作を伴う HTTPS 接続に対応していないようなので，IIOP/HTTPS での操作は失敗します。この状況に対応する方法はいくつかあります。
  - サーバーの証明書が，Internet Explorer がすでに信頼している CA の発行したものであることを確認してください。
  - 信頼された証明書として，このルート証明書を IE にインストールしてください。証明書ファイル（たとえば cacert.crt の bank_https など）を開くと，証明書をインストールすることができます。
  - ブラウザへルート証明書をダウンロードするには GateKeeper を使用してください。その方法については，次に説明します。
  - 商用 CA は通常，ルート証明書をインストールためのリンクを用意しています。
- デフォルトでは GateKeeper は，クライアント ID を確認しません。GateKeeper 設定ファイルで ssl_request_client_certificate を true に設定すると，この機能が有効になります。ところが，ユーザーの認証情報に応答する前にブラウザが権限を確認してしまうため，IIOP/HTTPS を使用できません。

Internet Explorer の場合，サーバー証明書の Common Name フィールドがサーバーのホスト名と同じでなければならないときもあります。[View | Internet Options] を選択し，[Advanced] タブを選択します。[Security セッション] までスクロールしてください。サーバーのホスト名を含まないサーバー証明書を使用するには，[Warn about invalid site certificates] チェックボックスのチェックをはずします。

第 3 章

認証

すべてのシステムのセキュリティ保護の最初の層は ID の提示および認証です。この層は，エンティティの身元を確認するプロセスを定義します。ほとんどの場合，エンティティの身元を確認するために認証情報が必要になります。

VisiSecure は，エンティティとシステムの対話に JAAS（Java Authentication and Authorization Service）フレームワークを使用します。同時に，セキュリティサブシステムの各種のコンポーネント間の認証情報の通信または転送のための*形式*（エンコードとデコードプロセス）を表すために認証メカニズムの概念が使用されます。

JAAS の基本概念

BSS（Borland Security Service）は，エンティティとシステムの対話に JAAS（Java Authentication and Authorization Service）フレームワークを使用します。JAAS を初めて使用する場合は，このサービスで使用する JAAS の用語を理解する必要があります。特に重要な用語として，*サブジェクト*，*プリンシパル*，および*認証情報*があります。

サブジェクト

JAAS で使用する*サブジェクト*という用語は，コンピュータサービスやリソースのユーザーを指します。ほかのコンピュータサービスやリソースを要求するコンピュータサービスやリソースもサブジェクトとみなされます。要求されたサービスやリソースは，名前でサブジェクトを認証します。ただし，サービスによっては，別の名前が使用される場合もあります。

たとえば，電子メールアカウントにはユーザー名とパスワードの組み合わせを使用し，ISP に対しては異なる組み合わせを使用するような場合です。ただし，どのサービスであっても認証を受けるサブジェクトは同じです。言い換えれば，1 つのサブジェクトに複数の名前を関連付けることができます。このように，サブジェクト自身が一定の時期におけるユーザー名やパスワードなどの認証メカニズムを知っている必要がある場合とは異なり，JAAS では 1 つのサブジェクトに対して複数の名前を関連付けてその情報を保持します。こうした複数の名前のそれぞれを*プリンシパル*と呼びます。

プリンシパル

プリンシパルは，サブジェクトに関連付けられている名前を表しています。1 つのサブジェクトは複数の名前を持つことができ，アクセスに必要な別名をサービスごとに持つこともできます。したがって，次のサンプルコードのように，サブジェクトは一連のプリンシパルで構成されています。

Java

```
public interface Principal {
   public String getName();
}
public final class Subject {
   public Set getPrincipals()
}
```

C++

```
class Principal {
    public:
       std::string getName() const=0;}
class Subject {
    public:
    Principal::set& getPrincipals();
}
```

サービスでサブジェクトが無事に認証されると，プリンシパルがサブジェクトに格納されます。強固なセキュリティを必要としない作業環境では，公開キーや証明書を使用する必要がありません。

アプリケーションコンテキストからサブジェクトのプリンシパル名を返すには，`getCallerPrincipal` を使用します。

メモ トランザクションに関係しているプリンシパルは，トランザクション内のプリンシパルの関連付けを変更することはできません。

認証情報

ほかのセキュリティ関連の属性をサブジェクトに関連付ける場合は，*認証情報*と呼ばれる JAAS の機能を使用します。認証情報は，パスワードや公開キー証明書と同じように汎用のセキュリティ関連属性で，どのようなオブジェクトでも認証情報にすることができるため，既存の証明書情報やインプリメンテーションを JAAS に移行することができます。認証データを別のサーバーまたはほかのハードウェアに保存する場合は，認証情報としてその認証データへのリファレンスを保存するだけです。たとえば，JAAS を使用すればセキュリティカードリーダーをサポートできます。

公開認証情報と秘密認証情報

JAAS の認証情報には，公開と秘密の 2 種類があります。*公開*認証情報はアクセス許可を必要としませんが，*秘密*認証情報ではセキュリティチェックが必要です。公開認証情報には公開キーなどを入れることができ，秘密認証情報には秘密キー，暗号化キー，機密パスワードなどを入れることができます。次のサブジェクトを見てください。

Java

```
public final class Subject {
   ...
   public Set getPublicCredentials()
}
```

C++

```
class Subject {
    public:
    Credential::set& getPrivateCredentials();
}
```

サブジェクトから公開認証情報を取り出す場合，次の場合を除いてアクセス許可は不要です。

Java
```
public final class Subject {
   ...
   public Set getPrivateCredentials()
}
```

C++
```
class Subject {
    public:
    Credential::set& getPrivateCredentials();
}
```

Java では，コードがサブジェクトの秘密認証情報にアクセスするためにアクセス許可が必要です。cpp では，すべてのコードはローカルなので信頼されます。公開認証情報および秘密認証情報のどちらにアクセスする場合もアクセス許可は不要です。Java のアクセス許可の詳細については，Sun Microsystems の JAAS 仕様を参照してください。

認証メカニズムと LoginModule

*認証メカニズム*は，セキュリティサブシステムの各種のコンポーネント間の認証情報の通信のエンコードとデコードの方法を表します。たとえば，27 ページの「LoginModules」がこのメカニズムと通信する方法，および 1 つのプロセスのメカニズムが別のプロセスの同等のメカニズムと通信する方法を表します。

VisiSecure には，サーバーとクライアントの認証に31 ページの「Borland LoginModules」，および Java と C++ の SPI（Security Provider Interface）のクラスが含まれます。このクラスによって，認証と識別のセキュリティサービスプロバイダのインプリメンテーションを「プラグイン」できます。

認証領域

*認証領域*は，ユーザー情報を含むデータソースをポイントするようにカスタマイズされたシングルユーザー認証メカニズムを表します。これにより，認証メカニズムが実際のユーザーデータベースから独立するので，同じ認証メカニズムをサポートする複数のユーザーデータベースが使用できるようになります。たとえば，LDAP を使用するように記述された認証モジュールは，別の環境の LDAP ディレクトリと対話できるので，認証メカニズムの書き直しや変更の必要はありません。

認証領域（ユーザードメイン）の詳細については，「セキュリティの概要」の31 ページの「Borland LoginModules」を参照してください。

LoginModules

LoginModule は認証メカニズムを定義し，特定の種類の認証メカニズムと対話するためのコードを提供します。各 LoginModule は特定のデータソースをポイントし，LoginModule の作成者が定義するその他のカスタマイズ可能な動作を提供する認証オプションを使ってカスタマイズできます。

各 LoginModule は特定の認証領域（NT ドメインなどの認証の本体つまり認証プロバイダ）を認証します。認証領域は，JAAS 設定ファイルの設定エントリによって表されます。JAAS 設定エントリには，領域を設定するためのオプションが関連付けられた 1 つ以上の LoginModule エントリが入っています。詳細については，29 ページの「LoginModule と領域の関連付け」を参照してください。

LoginContext クラスと LoginModule インターフェース

VisiSecure では，認証フレームワークのユーザー API として LoginContext クラスを使用します。LoginContext クラスは，JAAS 設定ファイルを使って現在のアプリケーションでどの認証サービスをプラグインするかを決定します。

Java

```
public final class LoginContext {
   public LoginContext(String name)
   public void login()
   public void logout()
   public Subject getSubject()
}
```

C++

```
class LoginContext{
    public:
        LoginContext(const std::string& name, Subject *subject=0,
CallbackHandler *handler=0);
        void login();
        void logout();
        Subject &getSubject() const;
}
```

認証サービス自体は，LoginModule インターフェースを使って関連する認証を実行します。

Java

```
public interface LoginModule {
   boolean login();
   boolean commit();
   boolean abort();
   boolean logout();
}
```

C++

```
class LoginModule {
    public:
        virtual bool login()=0;
        virtual bool logout()=0;
        virtual bool commit()=0;
        virtual bool abort()=0;
}
```

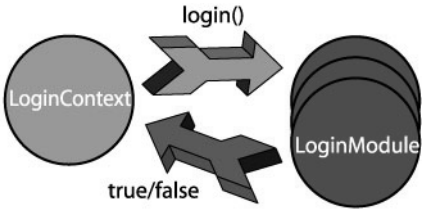
LoginModules をスタックして，1 つのサブジェクトを複数のサービスに対して同時に認証することもできます。

認証とスタックした LoginModule

スタックした LoginModules がすべて成功（または失敗）するように，認証は 2 つのフェーズに分けて実行されます。

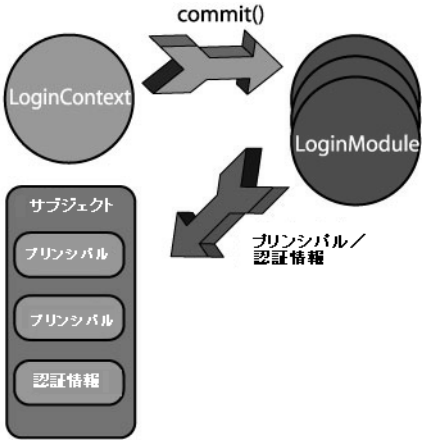
1 最初のフェーズは「ログインフェーズ」で，LoginContext が設定されているすべての LoginModules に対して login() を呼び出し，それぞれに認証を行います。

図 3.1　ログインフェーズ



2 必要な LoginModule のすべての認証が済むと，2 番めの「コミットフェーズ」が開始されます。ここでは，LoginContext が各 LoginModule に対して commit() を呼び出して認証を行います。このフェーズの間，LoginModules によって，作業の継続に必要な認証情報や認証済みプリンシパルがサブジェクトに格納されます。

図 3.2　コミットフェーズ



メモ　いずれかのフェーズが失敗すると，LoginContext は各 LoginModule に対して abort() を呼び出してすべての認証を終了します。

LoginModule と領域の関連付け

Borland VisiBroker Server では，JAAS 設定ファイルを使って LoginModule をその領域に関連付けて情報を格納します。JAAS 設定ファイルには，各認証領域のエントリが入っています。次に JAAS 設定エントリの例を示します。

```
MyLDAPRealm {
 com.borland.security.provider.authn.LDAPModule required URL=ldap://
directory.borland.com:389
}
```

次の図では，JAAS 設定ファイルにある領域エントリの要素を示しています。

図 3.3　JAAS 設定の領域エントリ

サーバーは複数の領域をサポートできます。これにより，クライアントは任意の領域の認証を受けることができます。サーバーは，それぞれの設定エントリを使ってサーバーを設定するだけで複数の領域をサポートできます。設定エントリの名前は定義されていないので，ユーザーが定義できます（例：PayrollDatabase）。

メモ　認証要件フラグ =required の場合は，少なくとも 1 つの LoginModule が必要です。

領域エントリの構文

各領域エントリは，それぞれの構文にしたがう必要があります。次のサンプルコードでは，ある領域エントリで使用している汎用的な構文を示しています。

```
//server-side realms for clients to authenticate against
realm-name {
   loginModule-class-name required|sufficient|requisite|optional
   [loginModule-properties];
   ...
};
```

メモ　LoginModule の各エントリの行末は，セミコロン（;）で表します。

この領域エントリには，次の 4 つの要素があります。

- **領域名** - 対応する LoginModule 設定が示す認証領域の論理名。
- **LoginModule Name** - 使用する LoginModule の完全修飾クラス名。
- **認証要件フラグ** - このフラグには，required，requisite，sufficient，および optional の 4 つの値があります。領域エントリの LoginModule ごとに，フラグ値を指定する必要があります。認証全体が成功するのは，required および requisite の LoginModule がすべて成功した場合だけです。sufficient LoginModule を設定して認証に成功した場合，sufficient LoginModule の前に記載された required および requisite の LoginModule が成功するだけで認証全体が成功します。アプリケーションに対して required または requisite のどちらの LoginModule も設定されていない場合，少なくとも sufficient または optional のいずれかの LoginModule が認証に成功する必要があります。この 4 つのフラグ値の定義は次のとおりです。

- **required** - 認証の成功に必要な LoginModule です。これが成功しても失敗しても，認証は各領域の LoginModule リストの下にある次のモジュールへと継続して実行されます。
- **requisite** - 認証の成功に必要な LoginModule です。この処理に成功すると，認証はこの領域の LoginModule リストの下にある次のモジュールへと継続して実行されます。失敗した場合は制御がただちにアプリケーションに返され，LoginModule のリストを順に認証していく処理は中断されます。
- **sufficient** - 認証の成功に必要ない LoginModule です。成功した場合は制御がすぐにアプリケーションに返され，LoginModule のリストを順に認証していく処理は中断されます。失敗した場合は，認証は LoginModule のリストの下にある次のモジュールへと継続して実行されます。
- **optional** - 認証の成功に必要ない LoginModule です。これが成功しても失敗しても，認証は LoginModule のリストの下にある次のモジュールへと継続して実行されます。

- **LoginModule 固有のプロパティ** - 場合によっては，サーバー管理者に各 LoginModule のプロパティを指定してもらう必要があります。次は，Borland が提供している各 LoginModule に必要なプロパティについて説明します。

Borland LoginModules

Borland は，サーバーとクライアントの認証でよく使用される次の LoginModule を提供します。これらの LoginModule は，クライアントの認証と，OS に対する 36 ページの「サーバーとクライアントの識別」の両方で使用します。

すべての LoginModule が同じプロパティを持つわけではありません。またユーザーの作成した LoginModule も異なるプロパティを持つ場合があります。VisiBroker に含まれる各 LoginModule およびその構文とプロパティについて説明し，領域エントリのサンプルコードも示します。

- Basic LoginModule - この LoginModule では，ユーザー情報の格納と取得に独自の方式を使用しています。標準 JDBC を使ってリレーショナルデータベースにデータを格納します。また，このモジュールは Tomcat JDBC 領域で使用されている独自方式にも対応しています。
- JDBC LoginModule - この LoginModule では，標準の JDBC データベースインターフェースを使用して，ネイティブデータベースのユーザーテーブルに対してユーザーを認証します。
- LDAP LoginModule - JDBC LoginModule とよく似ていますが，これは認証バックエンドとして LDAP を使用しています。
- Host LoginModule - サーバーをホストしている OS に対する認証で使用します。**これは，C++ 用にサポートされている唯一の LoginModule です。**

Basic LoginModule

この LoginModule では, 独自の方式でユーザー情報の格納と取得を行います。標準 JDBC を使ってリレーショナルデータベースにデータを格納します。また，このモジュールは Tomcat JDBC 領域で使用されている独自方式にも対応しています。

```
realm-name {
   com.borland.security.provider.authn.BasicLoginModule authentication-
requirements-flag
   DRIVER=driver-name
   URL=database-URL
   TYPE=basic|tomcat
   LOGINUSERID=user-name
   LOGINPASSWORD=password
   [USERTABLE=user-table-name]
   [GROUPTABLE=group-table-name]
   [GROUPNAMEFIELD=group-name-field-of-GROUPTABLE]
   [PASSWORDFIELD=field-name]
   [USERNAMEFIELDINUSERTABLE=field-name]
   [USERNAMEFIELDINGROUPTABLE=field-name]
   [DIGEST=digest-name]
};
```

ブラケット [ ] に囲まれている要素を使用するのは，Tomcat 領域への認証に必要な場合だけです。Tomcat 領域以外への認証には，残りのプロパティだけで十分です。

プロパティ	説明
DRIVER	パスワードバックエンドで使用するデータベースドライバの完全修飾クラス名です（例：com.borland.datastore.jdbc.DataStoreDriver）。
URL	この領域で使用するデータベースの完全修飾 URL。
TYPE	この領域で使用するスキーマ。この LoginModule は Tomcat JDBC 領域で使用するスキーマをサポートしており，このスキーマを使用できるようにします。Tomcat スキーマを使用するには，このプロパティを「TOMCAT」に設定します。Borland スキーマを使用するには，このプロパティを「basic」に設定します。**メモ：**このプロパティを「TOMCAT」に設定する場合は，ブラケット（[ ]）内にあるほかのプロパティもすべて設定する必要があります。
LOGINUSERID	パスワードバックエンドデータベースにアクセスするために必要なユーザー名。
LOGINPASSWORD	パスワードバックエンドデータベースにアクセスするために必要なパスワード。
[USERTABLE]	認証するユーザー名／パスワードという 2 つのエントリが格納されているテーブル名。
[USERNAMEFIELDINUSER-TABLE]	userID を読み出すことができる USERTABLE のフィールド名。
[USERNAMEFIELDIN-GROUPTABLE]	userID を読み出すことができる GROUPTABLE のフィールド名。USERTABLE のフィールド名とは異なります。
[PASSWORDFIELD]	認証するユーザー名のパスワードを含む USERTABLE のフィールド名。
[GROUPTABLE]	ユーザーのグループ情報を格納するテーブル名。TYPE を「TOMCAT」に設定すると，このテーブルのエントリが示す属性はグループとしてではなく，ロールとして扱われます。
[GROUPNAMEFIELD]	ユーザーに関連付けるグループ名を含んでいる GROUPTABLE のフィールド名。TYPE を「TOMCAT」に設定すると，このテーブルのエントリが示す属性はグループとしてではなく，ロールとして扱われます。
[DIGEST]	パスワードのダイジェストに使用するアルゴリズム。基本的な環境のデフォルトは SHA ですが，TYPE が「TOMCAT」に設定されている場合のデフォルトは MD5 です。

```
Premium {
   com.borland.security.provider.authn.BasicLoginModule required
   DRIVER="com.borland.datastore.jdbc.DataStoreDriver"
   URL="jdbc:borland:dslocal:/Security/java/prod/userauthinfo1.jds"
   Realm="Basic"
   LOGINUSERID="CreateTx"
   LOGINPASSWORD="";
};
```

パスワードはクリアテキストで保存すべきではないので，VisiSecure はパスワードを必ずダイジェストしてデータベースに保存します。digesttype オプションは，そのためのダイジェストアルゴリズムを定義します。デフォルトでは，基本タイプのスキーマには SHA アルゴリズムが使用され，「TOMCAT」タイプのスキーマには MD5 が使用されます。これは，digesttype オプションを設定することで変更できます。JVM が対応するダイジェストタイプのエンジンを見つけることができない場合は，かわりに SHA を使用します。SHA エンジンも見つからない場合，認証は常に失敗します。

JDBC LoginModule

この LoginModule では，標準 JDBC データベースインターフェースを使って認証を行います。

```
realm-name {
   com.borland.security.provider.authn.JDBCLoginModule authentication-
requirements-flag
   DRIVER=driver-name
   URL=database-URL
   [DBTYPE=type]
   USERTABLE=user-table-name
   USERNAMEFIELD=user-name-field-of-USERTABLE
   ROLETABLE=role-table-name
   ROLENAMEFIELD=field-name
   USERNAMEFIELDINROLETABLE=field-name
};
```

プロパティ	説明
DRIVER	領域で使用するデータベースドライバの完全修飾クラス名です（例：com.borland.datastore.jdbc.DataStoreDriver）。
URL	このパスワードバックエンドで使用するデータベースの完全修飾 URL。
[DBTYPE=ORACLE|SYBASE|SQLSERVER|INTERBASE]	サポートされているデータベースの種類。このオプションを指定するとテーブル情報は事前に設定されるため，指定する必要がなくなります。ただし，システムテーブルにアクセスできるように，ユーザー名／パスワードは指定する必要があります。
USERTABLE	データベースがユーザーを格納しているテーブル名。
USERNAMEFIELD	ユーザー名を含む USERTABLE のフィールド名。
ROLETABLE	データベースがユーザーの役割を格納しているテーブル名。
ROLENAMEFIELD	役割情報を格納している ROLETABLE のフィールド名。
USERNAMEFIELDINROLE-TABLE	ROLETABLE のユーザー名フィールドの名前。
USERNAME	パスワードバックエンドデータベースにアクセスするために必要なユーザー名。
PASSWORD	パスワードバックエンドデータベースにアクセスするために必要なパスワード。

```
LIMS {
   com.borland.security.provider.authn.JDBCLoginModule required
   DRIVER="com.borland.datastore.jdbc.DataStoreDriver"
   URL="jdbc:borland:dslocal:/Security/java/prod/userauthinfo.jds"
   USERTABLE=myUserTable
   USERNAMEFIELD=userNames
   ROLETABLE=myRoles
   ROLENAMEFIELD=roleNames
   USERNAMEFIELDINROLETABLE=userRole
   USERNAME="¥n"
   PASSWORD="¥n";
};
```

LDAP LoginModule

JDBC LoginModule とよく似ていますが，これは認証バックエンドとして LDAP を使用しています。

```
realm-name {
   com.borland.security.provider.authn.LDAPLoginModule authentication-
requirements-flag
   INITIALCONTEXTFACTORY=connection-factory-name
   PROVIDERURL=database-URL
   SEARCHBASE=search-start-point
   USERATTRIBUTES=attribute1, attribute2, ...
   USERNAMEATTRIBUTE=attribute
   QUERY=dynamic-query
};
```

プロパティ	説明
INITIALCONTEXTFACTORY	JNDI が LDAP にバインドするために使用する InitialContextFactory クラスです。
PROVIDERURL	LDAP サーバーの URL（ldap://<servername>:<port> 形式）です。
SEARCHBASE	ディレクトリのルックアップのための検索ベースです。
USERATTRIBUTES	このオプションは，指定されたユーザーに対して取得する属性を制御します。これは，認証されたユーザーに対して取得および保存される属性のカンマで区切られたリストです。属性は，ユーザーが指定された役割に属しているかどうかを判定するための承認ルールに使用できます。
USERNAMEATTRIBUTE	この属性は，ユーザーがユーザー名として入力する内容を表します。uid に設定すると，ユーザー名の入力を求められたときに uid を入力できます。mail に設定すると，ユーザー名の入力を求められたときに電子メールアドレスを入力できます。DN に設定すると，ユーザーは自身の認証のために完全な DN を入力します。

プロパティ	説明

QUERY	Query オプションは，LDAP に対して動的にクエリーを実行してほかの情報を取得し，結果を属性として表すためのメカニズムを提供します。たとえば，ユーザーは複数のグループのメンバーになっている場合があります。この情報を承認ドメインのルールで使用できるように GROUP 属性として抽出すると便利です。それには，クエリーを使用します。クエリーの形式は次のとおりです。
	`query.<suffix>="<attrname>=<ldap filter>";`
	suffix はこのエントリを一意に識別するために任意に指定し，クエリーはいくつでも指定できます。クエリーにユーザーの DN を挿入するには，{0} を使用する必要があります。LDAPLoginModule は，{0} をユーザーの実際の DN で置き換えます。たとえば，グループを照会して結果を GROUP 属性に保存するには，次のようにします。
	`query.1="GROUP=(&(ou=groups)(uniquemember={0}))";`
	これにより，ユーザーが所属し，uniquemember 属性にユーザーの DN が含まれるすべてのグループ（ou 属性の値が groups）が選択され，ユーザーの GROUP 属性の値として返されるオブジェクトの CN を保存します。指定される属性名が ROLE の場合，この属性の処理は JDBCLoginModule とまったく同じになります。この方法は，ユーザーの役割を LDAP に保存するために使用できます。

Host LoginModule

HostLoginModule は，UNIX や NT ベースのネットワークの認証で使用します。

```
realm-name {
   com.borland.security.provider.authn.HostLoginModule authentication-
requirements-flag;
};
```

Host LoginModule では，追加プロパティはまったく必要ありません。

```
Snoopy {
   com.borland.security.provider.authn.HostLoginModule required;
};
```

サーバーとクライアントの識別

さまざまな BES サービスに対して認証が必要な多くのクライアントとユーザーだけでなく，Borland VisiBroker Server 自体も ID を示す必要があります。セキュリティで保護されているほかのサーバーやサービスと通信する場合，Borland VisiBroker Server はこの ID を使って自己証明します。これにより，このサーバーがほかのクライアントのかわりに動作している場合，エンド層サーバーもこのサーバーの作成したアサーションを信頼することができます。一般に，セキュリティで保護されている通信にクライアントとして参加する必要があるすべてのシステムでは，そのシステムを動作しているユーザー／クライアントの ID を設定しておく必要があります。相互認証に SSL を使用する場合，サーバーはクライアントに身元を提示するためにも証明書を必要とします。

クライアント認証のための設定ファイルの設定

各処理は独自の設定ファイルを使用します。このファイルには，システムがクライアント認証でサポートしている認証領域に対する設定が含まれています。

設定ファイルの場所を設定するには

- vbroker.security.authentication.config プロパティを設定ファイルのパスに設定します。

システムの識別

セキュリティ設定は，プロパティと設定ファイルを使ってシステムを表す ID を定義します。この設定ファイルには，このプロセスが必要とする各領域への認証に必要なすべての LoginModule が含まれます。

次に例を示します。

vbroker.security.login=true と設定します。
vbroker.security.login.realms=payroll,hr と設定します。
vbroker.security.authentication.config=<config-file> によって，ファイルリファレンスに以下の領域情報を設定します。
vbroker.security.callbackhandler=<callback-handler> と設定します。

<config-file> に次の内容を設定します。

```
payroll {
com.borland.security.provider.authn.HostLoginModule required;
};

hr {
  com.borland.security.provider.authn.BasicLoginModule required
 DRIVER=com.borland.datastore.jdbc.DataStoreDriver
 URL="jdbc:borland:dslocal:../userdb.jds"
 TYPE=BASIC
 LOGINUSERID=admin
 LOGINPASSWORD=admin;
};
```

このサンプルコードでは，

- プロセスは，自身が認証を受ける必要がある領域について，vbroker.security.login.realms プロパティを通して何らかの情報を得ています。
- プロセスは，実行されている（論理的には「payroll」領域を示す）ホストに対して認証を実行し，この LoginModule を呼び出すように自身を設定することを認識しています。
- 「hr」領域にログインする必要があることも認識しているので，LoginModule にこのエンドを設定します。

vbroker.security.login.realms に渡される領域情報の形式は次のとおりです。

```
<authentication Mechanism>#<Authentication Target>
```

この形式は，*形式化ターゲット*と呼ばれます。

形式化ターゲット

「領域」は，認証元ターゲットの設定エントリを表します。設定ファイル（クライアントプロセス，または JAAS 設定ファイルに記載されていない証明書など）がない場合はターゲット領域を表す方法が必要です。これは，「形式化ターゲット」を使って行います。形式化ターゲットの形式は次のとおりです。

```
<authentication mechanism>#<mechanism specific target name>
```

次に例を示します。

```
Realm1, Realm3, GSSUP#Realm4, Certificate#ALL
```

認証メカニズムは，セキュリティサブシステムの各種のコンポーネント間の認証情報の通信の「形式」を表します。たとえば，LoginModule がこのメカニズムと通信する方法，および 1 つのプロセスのメカニズムが別のプロセスの同等のメカニズムと通信する方法を表します。個々のメカニズムのターゲット名は，メカニズムがこのターゲットを表す方法を示します。

GSSUP メカニズム

VisiSecure は，簡単なユーザー名／パスワード認証スキームのためのメカニズムを提供します。このメカニズムは，*GSSUP* と呼ばれます。OMG CSIv2 標準は，このメカニズムで相互使用できる形式を定義します。メカニズム対話モデルのための LoginModule は Borland によって定義されます。これは，このメカニズムのインプリメンテーションでは，LoginModule が提供する情報を CSIv2 を使って回線経由で転送できる情報（特定の形式）に変換する必要があるからです。

上記のように，ターゲット名はメカニズムによって異なります。GSSUP メカニズムのターゲット名は，ターゲット領域を表す簡単な文字列です（例：受信層の JAAS 設定ファイル）。したがって，サーバーに 1 つの領域が定義された設定ファイルがある場合（「ServerRealm」など），この領域のクライアント側の表現は次のとおりです。

```
GSSUP#ServerRealm
```

メモ GSSUP メカニズムは VisiBroker で常に使用できるので，便宜上，ターゲット名から「GSSUP」を省略できます。ただし，これは GSSUP メカニズムに対してだけです。セキュリティサービスが「領域」名を解釈するときは，まずローカルの JAAS 設定エントリを使って領域名の解決を試みます。これに失敗すると，次に領域名が「GSSUP」を表しているとして扱います。

証明書メカニズム

*証明書メカニズム*は，証明書による識別に使用するメカニズムです。このメカニズムは GSSUP とは異なり，ユーザー名とパスワードのかわりに証明書を使用し，ID は IIOP 層を介したより高いレベルの CSIv2 ではなく SSL 層を介して使用されます。

証明書は，証明書ログインまたは wallet API を使って VisiSecure に追加できます。wallet API を使用する場合は，vbsec.h ファイルの vbsec::WalletFactory クラスで定義する定数によって使用法を指定する必要があります。詳細については，「VisiSecure for C++ API」の class vbsec::WalletFactory を参照してください。

証明書ログインを使用する場合は，次の形式でターゲット領域を指定する必要があります。

```
Certificate#<target>
```

メモ 使用法を指定しない場合のデフォルトは ALL です。

ここでは，証明書ログインメカニズムに対して定義できるターゲットについて説明します。

表 3.1　証明書ログインメカニズムに対して定義されるターゲット

Target	説明
Certificate#CLIENT	このプロセスをクライアントロールで識別します。ユーザーがこのターゲットに対して ID を作成すると，作成された証明書 ID はこのプロセスがクライアントとして動作する際に使用されます。つまり，この証明書は SSL 送出接続を確立する際にこのプロセスを識別します。
Certificate#SERVER	このプロセスをサーバーロールで識別します。ユーザーがこのターゲットの ID を作成すると，このプロセスは SSL 接続を受け入れる際に身元確認のために，作成された証明書 ID を使用します。
Certificate#ALL	このプロセスをすべてのロールで識別します。この ID は，上記の両方のロールで使用されます。

プロセスはクライアントとサーバーで異なる ID を持つか，またはすべてのロールで使用される ID を持つことになります。つまり，Certificate#CLIENT と Certificate#ALL ターゲットに同時に ID を指定することはできません。

メモ　下位互換性がある場合は，wallet のプロパティと SSL API がサポートされてるので，このように確立される ID は Certificate#ALL としてのみ扱われます。

ボールトの使用

クライアントを実行する場合，セキュリティサブシステムはユーザーと対話して認証のための認証情報を取得します。これには，JAAS によって定義されるコールバックハンドラを使用します。ただし，サーバー（Visibroker サーバー，またはパーティション）を実行するときは，起動時にユーザーと対話することは好ましくないか，または不可能です。典型的な例としては，サーバーがホストの起動時にサービスとして起動されるか，または一種の自動化されたスクリプトから起動される場合があります。

*ボールト*は，このような環境でセキュリティサブシステムに識別情報を提供するように設計されています。ボールト自身がセキュリティサブシステムに直接連結されていないことに注意してください。ボールトは，単にユーザーとの対話を代行するためのツールです。つまり，ボールトは認証された認証情報を持っていません。セキュリティサービスは，適切なすべての認証を実行しますが，コールバックハンドラと対話するかわりにボールトから情報を受け取ります。ユーザーとの対話が必要ないので，ボールトのデータが十分に安全であれば，機密情報（ユーザー名とパスワード）が入っていることがあります。したがって，そのようなサーバーの認証に使用するボールトファイルは，ファイルアクセス許可などのホストセキュリティメカニズムまたはその他の同等の方法によって保護する必要があります。

ボールトの作成

ボールトを作成するには，それぞれのインストールの bin ディレクトリから vaultgen コマンドラインツールを使用します。用法は次のとおりです。

```
vaultgen [<driver-options>] -config <config.jaas-file> -vault <vault-name>
[<options>] <command>
```

<driver-options> はオプションで，次のいずれかです。

- -J<option> : -J Java オプションを直接 JVM に渡します。
- -VBJVersion : VBJ バージョン情報を出力します。
- -VBJDebug : VBJ デバッグ情報を出力します。
- -VBJClasspath : CLASSPATH 環境変数の前に置くクラスパスを指定します。
- -VBJProp <name=value> : 名前／値のペアを VM に渡します。
- -VBJjavavm : Java VM にパスを指定します。
- -VBJaddJar <jar-file> : VM を実行する前に JAR ファイルを CLASSPATH に追加します。

-config <config.jaas-file> は，ボールトの ID が認証を受ける領域が入っている config.jaas ファイルの場所をポイントします。-vault <vault-name> は，生成されるボールトのパスです。既存のボールトを指定してそれに ID を追加することもできます。

<options> はその他のオプションの引数で，次のいずれかです。

- -?, -h, -help, -usage：使用情報を出力します。
- -driverusage：ドライバオプションを含む使用情報を出力します。
- -interactive：対話型のシェルを有効にします。

<command> は，vaultgen に実行させるコマンドです。次のいずれかを選択できます。

- login <realm|formatted-target>：指定された領域または形式化ターゲットのためのボールトの ID を作成します。ID は，システム起動のログイン時にボールトが使用されるときに初めて作成されます。
- logout <realm|formatted-target>：指定された領域または形式化ターゲットのためのボールトの ID を削除します。
- runas <alias> <realm>：指定された領域に提供された ID を使って run-as エリアスを設定します。
- removealias <alias>：ボールトから設定された run-as エリアスを削除します。
- realms：この設定で使用できる領域を一覧表示します。
- mechanisms：この設定で使用できる形式化ターゲットのためのメカニズムを一覧表示します。
- aliases：ボールトに設定されたエリアスを一覧表示します。
- identities：ボールトに設定された ID を一覧表示します。

VaultGen の例

次に，VaultGen に関する例を示します。ここでは，base というドメインで使用する MyVault というボールトを作成します。まず，ドメインが使用するセキュリティプロファイルを特定し，config.jaas ファイルを参照する必要があります。ドメインの orb.properties ファイルにある vbroker.security.profile プロパティの値を確認します。

```
#
# ユーザードメインのセキュリティ
#
# デフォルトでユーザードメインのセキュリティを無効化する
vbroker.security.profile=default
vbroker.security.vault=${properties.file.path}/../security/scu_vault
```

セキュリティプロファイルの名前は default です。つまり，プロファイルの config.jaas ファイルへのパスは次のとおりです。

```
c:/BDP/var/security/profiles/default/config.jaas
```

ここで，ID を作成するプロファイルに記載されている領域を確認できます。インストール先の bin ディレクトリに移動し，次のように realms コマンドを使用します。

```
c:¥BDP¥bin> vaultgen -config ../var/security/profiles/default/config.jaas -
vault myVault realms
```

vaultgen によって，次の領域を使用できることがわかります。

```
The following realms are available:
- UserRealm
- MikeRealm
- BenRealm
```

次に，login コマンドを使って vaultgen を実行します。

```
c:¥BDP¥bin> vaultgen -config ../var/security/profiles/default/config.jaas -
vault myVault login UserRealm
```

vaultgen は UserRealm のユーザー名とパスワードの入力を要求するので，それらを入力します。それぞれの領域に対してこのプロセスを繰り返します。各コマンドの実行後に，vaultgen は新しい ID でログインし，MyRealm に変更を保存したことを通知します。

```
Logged into realm BenRealm
Generating Vault to MyVault
```

ボールトはコマンドで指定するディレクトリに作成され，この場合は bin ディレクトリです。実際のボールトファイルの保存に適した場所は，次の場所にあるドメインの security ディレクトリです。

```
<install-dir>/var/domains/<domain-name>/adm/security/
```

クライアントの識別

上記のサンプルとは異なり，認証の必要がある領域の情報をクライアントプロセスが持っていない場合もあります。この場合，クライアントはデフォルトで使用可能な領域リストをサーバーの IOR に問い合わせ，ユーザー名とパスワードを提供する領域を選択するオプションをユーザーに提示します。サーバーはこのユーザー名とパスワードを使用し，指定された領域の設定ファイルを確認し，クライアントから収集した情報を指定された LoginModule のためのデータとして使用します。

たとえば，次のようなサーバー側設定ファイルでは，収集された情報またはユーザーが入力した情報が JDBCLoginModule に使用されます。

```
SecureRealm{
   com.borland.security.provider.authn.JDBCLoginModule required
   DRIVER=""com.borland.datastore.jdbc.DataStoreDriver"
   URL="jdbc:borland:dslocal:../userdb.jds"
   USERNAMEFIELD="USERNAME"
   GROUPNAMEFIELD="GROUPNAME"
   GROUPTABLE="UserGroupTable"
};
```

プロセスのデフォルトの動作は，プロパティを使って変更できます。vbroker.security.authentication.retryCount を設定することで，再試行回数を設定できます。デフォルトは 3 です。認証のためのセキュリティプロパティを含むセキュリティプロパティの一覧および説明については，第 9 章「セキュリティプロパティ（Java）」と第 10 章「セキュリティプロパティ（C++）」を参照してください。

第 4 章

承認

*承認*は，サーバーにおいて要求した操作を実行する権限をユーザーが持っているかどうかを確認するプロセスです。たとえば，クライアントがエンタープライズ Bean メソッドにアクセスする場合，アプリケーションサーバーはクライアントユーザーがアクセスする権限を持っているかどうかを確認する必要があります。承認は，認証（ユーザーの ID の確認）の後に行われます。

承認はユーザー ID とアクセスコントロールリスト（ACL）に基づいて行います。ACL とは，指定の機能にアクセスできる権限のあるユーザーのリストのことです。通常，アクセスコントロールリストには特別なリソースを使用できる一連のロールを指定します。また，属性が特定のロールに合ったユーザーを指定することで，そのロールのアクションを実行できるユーザーを指定することができます。

Borland では，ロールに基づいたアクセスコントロール方式を使用します。配布デスクリプタには，各エンタープライズ Bean メソッドにアクセスするための承認を得ているロールのリストが格納されます。Borland Security Service は，ロールデータベース（ロール DB）を使ってユーザー ID を EJB ロールに関連付けます。ユーザーに少なくとも 1 つのロールが許可されて関連付けられていると，メソッドへアクセスできます。

ロール DB を使ったアクセスコントロールの定義

ロール DB はテキストファイルで，ロールと，ロールに関連付けられているアクセス ID が格納されています。ロール DB のロールごとに，*ロールエントリ*を構成しています。

VisiBroker のロール DB ファイルは，セキュリティプロファイルとともに次の Borland Deployment Platform のインストール領域にあります。

```
<install-dir>/var/security/profiles/<profile-name>/
```

デフォルトのロール DB（`default.rolemap`）は次の場所にあります。

```
<install-dir>/var/security/profiles/default/default.rolemap
```

VisiBroker のロールマップファイルの場所は，`vbroker.security.domain.<authorization-domain>.rolemap_path` プロパティによって指定されます。

ロール DB ファイルは，プリンシパル（クライアント ID）のアクセス権を判定するために使用します。ロール DB に定義される各ロールには，クライアント ID が割り当てられます。アクセス権は，個々のクライアント ID ではなくロールに基づいて与えられます。たとえば，アプリケーションが Sales Clerk ロールを承認する場合は，すべての販売員のユー

ザー ID を Sales Clerk ロールに割り当てます。
その後に Sales Clerk ロールに add_purchase_order メソッドなどの特定の操作を実行するための権利を認可します。これにより，Sales Clerk ロールに関連付けられた販売担当者全員が add_purchase_order を実行できるようになります。

ロール DB の構造

ロール DB ファイル自体も次のような形式で，複数のロールエントリを指定することができます。

```
role-name {
   assertion1 [, assertion2, ... ]
   ...
   [assertion-n]
   ...
}
role-name2 {
   assertion3 [, assertion4, ... ]
   ...
   [assertion-n]
   ...
}
```

ロールエントリは，ロール名と，中括弧（{ }）で囲んだルールのリストから構成されています。ロールには，1 つまたは複数のルールを設定する必要があります。各*ルール*は 1 行で構成され，適切なアクセス ID を表すカンマで区切られた*アサーション*のリストが入っています。つまり，各ルールには 1 つまたは複数のアサーションを指定する必要があります。

ロールエントリの各行が*ルール*です。ルールは 1 つずつ上から順に読み込まれ，適合するルールが見つかるか，適合するルールがないと判断されるまで承認処理が行われます。これは，各ルールが OR 演算子で区切られている場合と同じように読み込まれるためです。*アサーション*は 1 つの行にあり，カンマ（,）で区切られています。アサーションは左から右へ順に読み込まれ，ルールが成功するにはすべてのアサーションが成功する必要があります。これは，ルール中の各アサーションが AND 演算子で区切られている場合と同じように読み込まれるためです。

各ルールには，指定したプリンシパルのセキュリティ認証情報に必要なセキュリティ情報をすべて指定する必要があります。つまり，各プリンシパルにはルールに必要な最小限の属性，または記載されているすべての属性が必要です。余分な属性を指定すると，承認は失敗します。

アサーションの構文

論理演算子と，承認に必要なアクセス ID を示す属性／値のペアを一緒に使ってルールを指定するには，さまざまな方法があります。また，ワイルドカード（*）を使用したシンプルな構文にしたがってルールをさらに柔軟なものにする方法もあります。この両方の構文を次に説明します。

アサーションと論理演算子の使い方

属性／値のペアを指定する場合，2 つの論理演算子が利用できます。

表 4.1　承認アサーションのための論理演算子

演算子	説明	サンプル
属性 **=** 値	**equals**：承認を成功させるには，この属性は値と同じにする必要があります。	CN=Russ Simmons
属性 **!=** 値	**not equal**：承認を成功させるには，属性がルールの値と異なっている必要があります。	CN!=Rick Farber

どのような文字列も値にできますが，ワイルドカード（*）は特殊な用途向けです。たとえば，属性／値のペアの GROUP=* はすべての GROUP に一致します。次のロールには 2 つのルールが関連付けられています。

```
manager {
   CN=Kitty, GROUP=*
   GROUP=SalesForce1, CN=*
}
```

ロールマネージャには 2 つのルールが関連付けられています。1 番めのルールでは，Kitty という名前のユーザーは，承認時に関連付けられているグループに関係なく，manager として承認されます。2 番めのルールでは，グループ SalesForce1 に属するユーザーはだれでも，common-name（CN）に関係なく承認されます。

ワイルドカードアサーション

複雑なセキュリティ階層になると，プリンシパルを承認するために 1 つまたは 2 つの属性だけでは不安な場合があります。Borland のセキュリティ階層は，最上位が GROUP，次に ORGANIZATION（O）と ORGANIZATIONAL UNIT（OU）に分岐し，最後は COMMON NAME（CN）になります。

たとえば，販売業務のマネージャにメソッド許可を与える SalesSupervisor というセキュリティロールを定義するとします（例：「sales」が ORGANIZATION で，「managers」が ORGANIZATIONAL UNIT）。こうした操作を実行するには，次のルールを使用します。

```
SalesSupervisor {
   GROUP=*, O=sales, OU=managers, CN=*
}
```

ここでは，GROUP の値も COMMON NAME の値も指定していません（不要と思われるため）。ただし，各ルールには，プリンシパルの認証情報が取りうる値をすべて指定する必要があります。*ワイルドカードアサーション*を使用して，少ないスペースで同じ情報を表すこともできます。

ワイルドカード（*）をアサーションの直前に置くと，ワイルドカードアサーションを作成することができます。ワイルドカードを 1 つのアサーションの前に置くということは，取りうるセキュリティ属性はどれも受け入れられるが，このセキュリティ属性には 1 つのアサーションが存在している*必要がある*ことを意味しています。もう 1 つの方法では，括弧で囲んだカンマ区切りのアサーションリストの前に，ワイルドカードを置きます。これは，取りうるセキュリティ属性はどれも受け入れられるが，セキュリティ属性には，括弧の中にリストされたアサーションが*必ず*入っていることを示しています。

ワイルドカードアサーションを使用すると，ロールは次のようになります。

```
SalesSupervisor {
   *O=sales, *OU=managers
}
```

さらにシンプルなコードにすることもできます。

```
SalesSupervisor {
   *(O=sales, OU=managers)
}
```

この 3 つのサンプルコードは，すべて同じルールを使ったバリエーションです。

その他のアサーション

あるロールによって，ほかのロールの拡張性は制限を受けます。ロールエントリの一部として，role=existing-role-name アサーションを指定して，これまでのロールを拡張することができます。
また，ロール DB 構文のかわりに，承認メカニズムとして第 11 章「VisiSecure for C++ API」によるカスタムコードを使用することもできます。

既存のロールの再利用

ルールリファレンスアサーション（role=role-name）を使用すると，既存のロールからルールを参照できます。たとえば，セールス主任を兼任するマーケティング担当者のグループが同じコードで Sales Supervisors として承認を受けるとします。SalesSupervisor サンプルコードを使用すれば，次のように新しいロールエントリを作成できます。

```
MarketSales {
   role=SalesSupervisor
   *(OU=marketing)
}
```

これで，SalesSupervisor というロールに属する全員が「marketing」OU のスタッフと同様に，MarketSales というロールを利用できるようになります。

承認ドメイン

それぞれのロール DB ファイルは，*承認ドメイン*に関連付けられています。承認ドメインは，ロール DB およびその承認許可を区切るために使用するセキュリティコンテキストです。基本セキュリティモデルにおける承認ドメインの詳細については，第 2 章「セキュリティの概要」を参照してください。

EJB は，さまざまな許可とロールを持っている複数のセキュリティコンテキストに配布できます。

メモ　承認ドメインは，配布デスクリプタで EJB に関連付けられます。

承認ドメインは VisiBroker ORB に登録されている限り，いくつでも使用することができます。各承認ドメインに対して，次の操作を実行する必要があります。

- 名前を付ける
- デフォルトアクセスを設定する
- ロール DB を設定する
- エリアスを設定する

それには，次のプロパティを設定します。次のプロパティの詳細については，第 9 章「セキュリティプロパティ（Java）」または第 10 章「セキュリティプロパティ（C++）」を参照してください。

表 4.2　承認のための ORB プロパティ

プロパティ	説明
vbroker.security.authDomains=<domain1>[, <domain2>, <domain3>, ...]	承認ドメイン名のリスト
vbroker.security.domain.<domain-name>.defaultAccessRole=grant | deny	<domain-name> がない場合にデフォルトでドメインへのアクセスを許可するかどうかを設定します。
vbroker.security.domain.<domain-name>.rolemap_path=<path>	承認ドメイン *domain-name* に関連するロール DB ファイルのパスです。これは相対パスでも指定できますが，Borland ではフルパス名での指定をお勧めします。
vbroker.security.domain.<domain-name>.runas.<role-name>=<alias> | use-caller-identity	このプロパティを使って run-as ロール（<role-name>）の ID をセットアップします。alias は，ボールトの使用の alias を表します。呼び出し元プリンシパル自身を run-as ロールのプリンシパル ID として使用するには，use-caller-identity を使用します。

run-as エリアス

メモ C++ では，run-as エリアスは使用できません。

run-as エリアスは認証 ID を識別する文字列です。run-as エリアスはボールトで定義され，VisiBroker ORB 内にスコープされます。このエリアスは，特定のユーザーを表します。ID は，class vbsec::Context を使用するか，またはボールトで定義することによってエリアスにマッピングされます。ボールトには，run-as エリアスおよびそれに対応する認証用の認証情報のリストを入れることができます。認証情報は ID が代理で実行するために使用します。どちらの場合も，認証のための認証情報（ボールトまたはウォレットから取得）は LoginModule に渡され，LoginModule によって認証され，run-as マップの認証情報に対応する完全に認証された ID として設定されます。

承認ドメインは，ドメインのロールのエリアスとして実行できるように設定されます。要求が指定されたロールで実行できるようになると，そのコンテキストの承認ドメインは，対応する run-as エリアスを取得するように求められます。run-as マップを参照することによってエリアスに対応する ID が取得され，その ID が使用されます。

run-as の ID は，ユーザー名とパスワードの ID だけでなく，証明書 ID に設定することもできます。

run-as マッピング

メモ C++ では，run-as マッピングは使用できません。

vbroker.security.domain.<domain-name>.runas.<role-name> プロパティを設定すると，エリアスを効果的に Bean の run-as ロールにマッピングできます。承認に成功してからメソッドを呼び出すまでに，コンテナはメソッドの EJB 配布デスクリプタに指定されている run-as ロールをチェックします。run-as ロールが存在する場合，エリアスがあるかどうかも確認します。エリアスも存在する場合，Bean が外部呼び出しを行うときに，そのエリアスの ID に切り替えます。

ただし，エリアスが指定されていない場合（つまり，run-as ロール名が use-caller-identity に設定されている場合）は，呼び出し元のプリンシパル名が使用されます。

CORBA 承認

CORBA 環境での承認は，指定オブジェクトの指定ロールにある ID しか，そのオブジェクトにアクセスできません。オブジェクトのアクセスポリシーは，問題オブジェクトのホストであるポータブルオブジェクトアダプタ（POA）の，保護品質ポリシーで指定されています。アクセスポリシーを適用できるのは，POA レベルにだけです。

CORBA オブジェクトの承認を実装する場合，ロールマップも使用できます。同様に，J2EE ロールとその概念も CORBA 環境で使用できます。

CORBA オブジェクトの承認の設定

オブジェクトに承認を設定するには，次の手順を実行します。

1 ServerQopPolicy を作成する。

2 ServerQopConfig オブジェクトを使って ServerQopPolicy を初期化します。

3 次のように AccessPolicyManager インターフェースを実装します。

Java
```
interface AccessPolicyManager {
   public java.lang.String domain();
   public com.borland.security.csiv2.ObjectAccessPolicy getAccessPolicy(
      org.omg.PortableServer.Servant servant, byte[] object_id byte []
adapter_id);
}
```

C++
```
class AccessPolicyManager {
    public:
        virtual char* domain() =0;
        ObjectAccessPolicy_ptr getAccessPolicy(PortableServer_ServantBase*
_servant,
        const ::PortableServer::ObjectId& id,
        const::CORBA::OctetSequence& _adapter_id) =0;
}
```

このインターフェースは，domain() メソッドから承認ドメインを返し，この承認ドメインを使って ServerQopConfig オブジェクトのアクセスマネージャを設定します。承認ドメインには，適切なロールマップに関連付けられている承認ドメイン名を指定します。次のようにプロパティを設定して，ロールマップの場所と名前を設定できます。

```
vbroker.security.domain.<authorization-domain-name>.<rolemap-path>
```

ここで，<authorization-domain-name> が重複するため，<rolemap-path> はロールマップファイルを基準とした相対パスにします。getAccessPolicy() メソッドは，サーバントのインスタンス，オブジェクト ID，およびアダプタ ID を受け取り，ObjectAccessPolicy インターフェースのインプリメンテーションを返します。

1 必須のロールと，オブジェクトメソッドへのアクセスに必要な run-as ロールを返す ObjectAccessPolicy インターフェースを実装します。Borland のインプリメンテーションでは，J2EE の run-as ロールと CORBA の run-as ロールとの間に違いはありません。ObjectAccessPolicy インターフェースは，次のとおりです。

Java
```
interface ObjectAccessPolicy {
    public java.lang.String[] getRequiredRoles(java.lang.String method);
    public java.lang,String getRunAsRole(java.lang.String method);
}
```

C++
```
class ObjectAccessPolicy {
    public:
        getRequiredRoles (const char* _method) =0;
}
```

getRequiredRoles() メソッドは引数としてメソッド名を受け取り，ロールのシーケンスを返します。getRunAsRole() メソッドは，メソッドのアクセスに必要な run-as ロールがある場合はこのロールを返します。

ID は，コールバックハンドラを使って指定します。詳細については，第 3 章「認証」を参照してください。

第 5 章

ドメインのセキュリティプロファイルの設定

Borland VisiBroker がインストールされている場合，システムの各ドメインに対して関連する一連のセキュリティパラメータを定義できます。プロファイルを有効にするには，該当するドメインプロパティを設定する必要があります。各プロファイルは，適切なセキュリティプロパティ，ロールマップ，および設定ファイルによって設定します。ここでは，セキュリティプロファイルを設定し，VisiSecure がアプリケーションのセキュリティを保護するために必要なデータを提供する方法について説明します。

セキュリティプロファイル

VisiSecure では，*プロファイル*というセキュリティ情報のリポジトリを設定できます。プロファイルには，認証用の 第 3 章「認証」を定義するための config.jaas ファイルおよび認証ロールマップ 第 4 章「承認」が入っています。プロファイルにユーザーデータベースおよびスクリプトファイルが入っている場合もあります。

プロファイルはインストール場所の security ディレクトリの次の場所で定義されます。

```
<install_dir>/var/security/profiles/<profile_name>
```

各プロファイルには一意の名前を付けることができます。<profile_name> ディレクトリには，少なくとも次のファイルが必要です。

- config.jaas：LoginModule を定義する JAAS 認証設定ファイル。
- <role_db>.rolemap：承認データが入っている Role DB ファイル。
- security.properties：VisiSecure サービスの動作を定義するセキュリティプロパティの中央のリポジトリ。

プロファイルは，管理コンソールを使って表示および設定することもできます。管理コンソールを使ってセキュリティプロファイルを表示するには，次の手順にしたがいます。

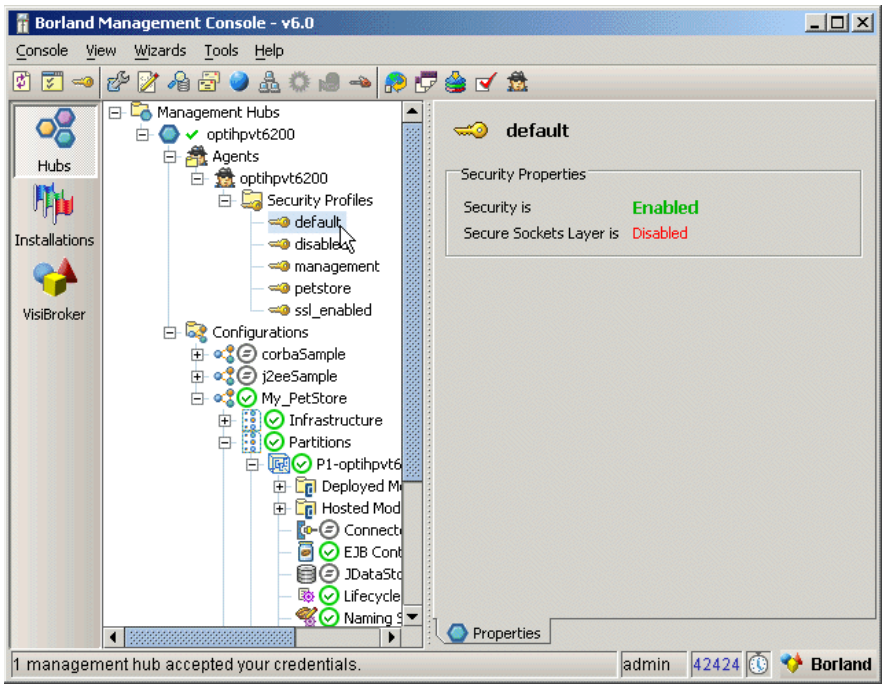
1 ［Hubs］ビューで［Management Hubs］ノードを展開します。

2 ［Agents］ノードを展開します。

3 セキュリティプロファイルを表示するドメインのエージェントを選択します。

4 個々のセキュリティプロファイルを選択します。

図 5.1　セキュリティプロファイルの選択



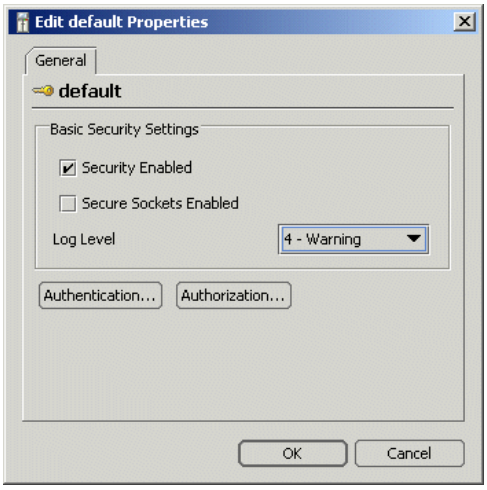
セキュリティの有効化

ドメインをセキュリティで保護するには，有効セキュリティプロファイルを関連付ける必要があります。セキュリティプロファイルを有効にするには，次の手順にしたがいます。

1 ［Hubs］ビューで，編集するプロファイルに移動します。

2 プロファイルを右クリックし，コンテキストメニューから［Configure］を選択します。［Edit Default Properties］ダイアログが表示されます。

図 5.2　［Edit Default Properties］ダイアログ



3 ［Security Enabled］チェックボックスをチェックします。

4 ［OK］をクリックします。

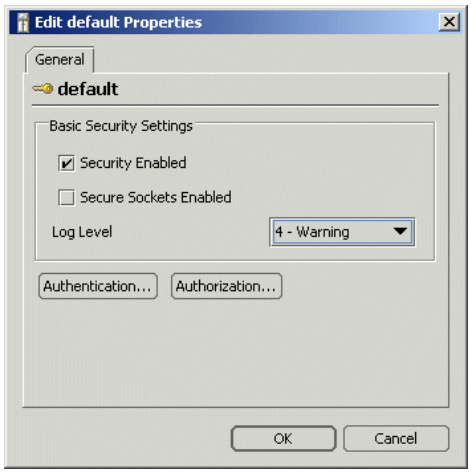
SSL の有効化

SSL を使用するには，セキュリティプロファイルで SSL が有効にする必要があります。プロファイルの SSL を有効にするには，次の手順にしたがいます。

1 ［Hubs］ビューで，編集するプロファイルに移動します。

2 プロファイルを右クリックし，コンテキストメニューから［Configure］を選択します。［Edit Default Properties］ダイアログが表示されます。

図 5.3　プロファイルの SSL の有効化



3 ［Secure Sockets Enabled］チェックボックスをチェックします。

4 ［OK］をクリックします。

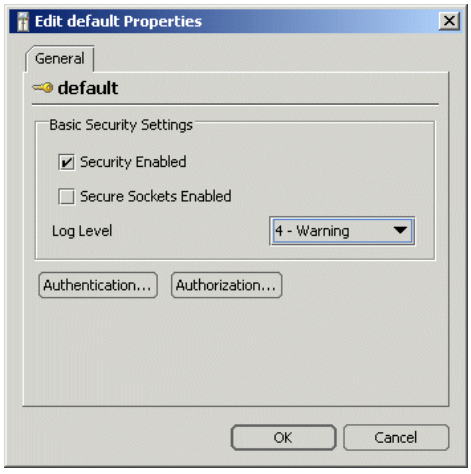
ログレベルの設定

セキュリティプロファイルのログレベルを設定するには，次の手順にしたがいます。

1 ［Hubs］ビューで，編集するプロファイルに移動します。

2 プロファイルを右クリックし，コンテキストメニューから［Configure］を選択します。［Edit Default Properties］ダイアログが表示されます。

図 5.4　セキュリティプロファイルのログレベルの設定



3 ［Log Level］ドロップダウンリストから目的のログレベルを選択します。

4 ［OK］をクリックします。

認証の設定

プロファイルの認証を設定するには，config.jaas ファイルを手作業で作成するか，または Borland 管理コンソールを使用します。

config.jaas ファイルの作成

config.jaas ファイルには，ユーザーを 1 つ以上の領域に認証したり，認証メカニズムを定義したり，特定の種類の認証メカニズムと対話するためのコードを提供するために必要なデータが入っています。たとえば，config.jaas ファイルは次のようになります。

```
UserRealm {
  com.borland.security.provider.authn.BasicLoginModule required
 DRIVER=com.borland.datastore.jdbc.DataStoreDriver
 URL="jdbc:borland:dslocal:${config.jaas.path}/userdb.jds"
 TYPE=BASIC
 LOGINUSERID=admin
 LOGINPASSWORD=admin;
};
```

これにより，BasicLoginModule を使用する必要がある UserRealm という領域が定義されます。使用するデータベースおよびログインの方法に関する情報も提供します。LoginModule およびそのオプション，このような領域エントリの構文の詳細については，第 3 章「認証」を参照してください。

完成した config.jaas ファイルは，プロファイルのフォルダに保存されます。

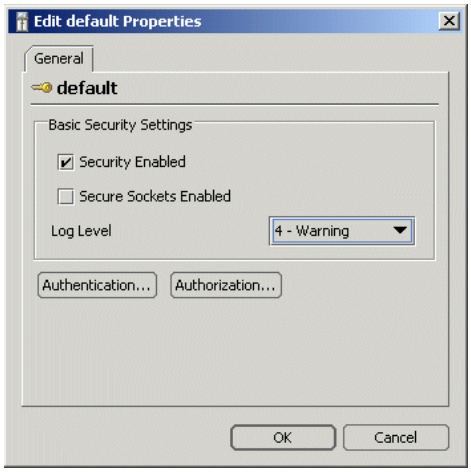
管理コンソールによる認証の設定

管理コンソールを使って認証を設定するには，次の手順にしたがいます。

1 ［Hubs］ビューで，編集するプロファイルに移動します。

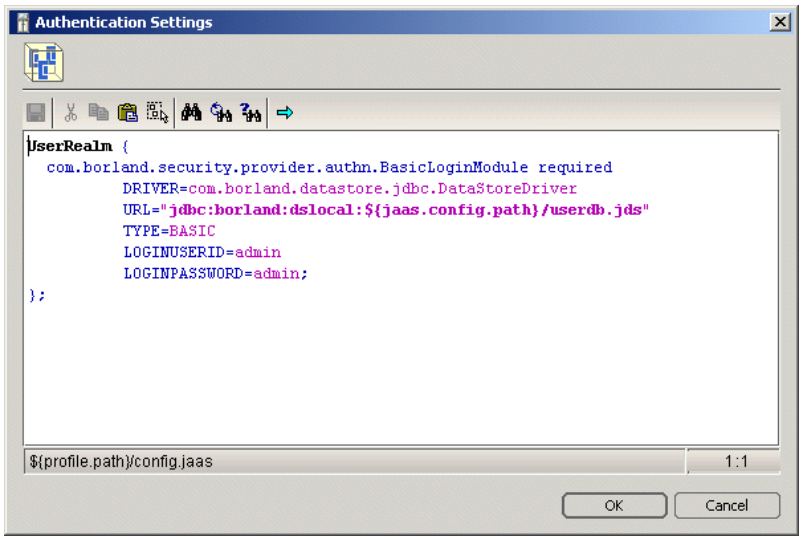
2 プロファイルを右クリックし，コンテキストメニューから［Configure］を選択します。［Edit Default Properties］ダイアログが表示されます。

図 5.5　認証の設定



3 ［Authentication］をクリックします。［Authentication Settings］ダイアログが表示されます。

図 5.6 ［Authentication Settings］ダイアログ



4 ［Authentication Settings］ダイアログの編集ウィンドウに，プロファイルの config.jaas ファイルの内容が表示されます。領域エントリを編集または追加します。詳細については，第 3 章「認証」を参照してください。

5 終了したら，［OK］をクリックします。

承認の設定

承認を設定するには，独自の承認ロールマップを手作業で作成するか，または管理コンソールを使用します。

ロールマップファイルについて

承認ロールマップは，.rolemap ファイルにキャプチャされます。一般に，このファイルは承認ドメインに基づいて名前を付けますが，必須事項ではありません（たとえば，"default" というプロファイルのロールマップは default.rolemap になります）。ロールマップファイルは *Role DB* とも呼ばれ，ロールへのユーザーのマップまたは*アクセスコントロールリスト*です。通常，アクセスコントロールリストには特別なリソースを使用できる一連のロールを指定します。ロールマップは，属性が特定のロールに合ったユーザーを指定することで，そのロールを実行できるユーザーを指定することができます。

VisiSecure は，ロール名，およびロールを定義する一連の属性を指定するためのメカニズムを提供します。たとえば，Role DB のコンテンツは次のとおりです。

```
ServerAdministrator {
  CN=*, OU=Security, O=Borland, L=San Mateo, S=California, C=US
  *(CN=admin)
  *(GROUP=administrators)
}

Customer {
  role=ServerAdministrator
  *(CN=borland)
  *(CN=pclare)
  *(CN=jeeves)
  *(GROUP=RegularUsers)
}
```

これにより，ServerAdministrator と Customer の 2 つのロールが定義され，それらを規定する一連のルールと属性も定義されます。ロールを定義し，顧客のロールマップを記述する方法については，第 4 章「承認」を参照してください。

完成したロールマップファイルは，config.jaas ファイルとともにプロファイルのフォルダに保存されます。

管理コンソールによる承認の設定

［Authorization Settings］ダイアログを使用すれば，セキュリティプロファイルの承認を設定できます。これにより，次のことができます。

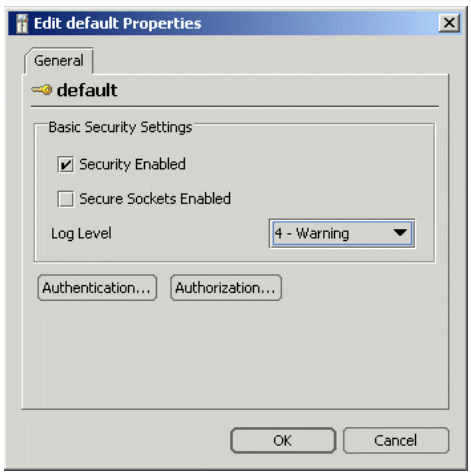
- 承認ロールマップとルールの表示。
- ドメインの承認ロールマップの追加，編集，および削除。
- 承認ロールマップ内のロールの追加，編集，および削除。
- ロール内のルールの追加，編集，および削除。

［Authorization Settings］ダイアログにアクセスするには，次の手順にしたがいます。

1 ［Hubs］ビューで，編集するプロファイルに移動します。

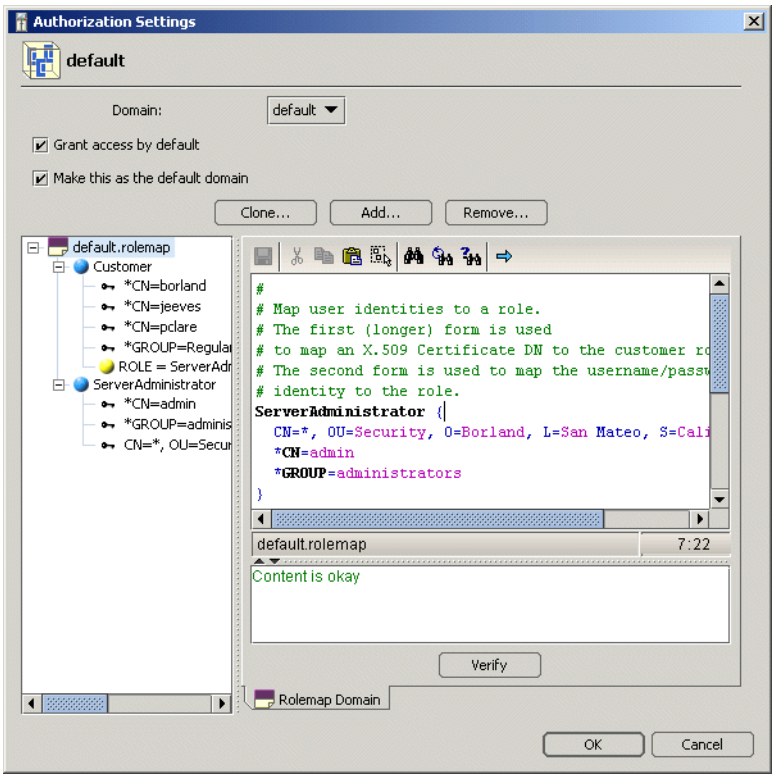
2 プロファイルを右クリックし，コンテキストメニューから［Configure］を選択します。［Edit Default Properties］ダイアログが表示されます。

図 5.7　承認設定ダイアログへのアクセス



3 ［Authorization］をクリックします。［Authorization Settings］ダイアログが表示されます。

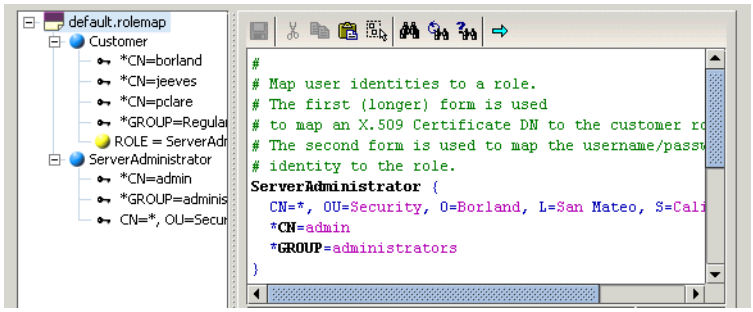
図 5.8　承認設定



[Authorization Settings] ダイアログの左側のペインには，選択した承認ドメインのロールおよびそれに関連付けられているルールのツリービューが表示されます。ツリーのノードを選択すると，ノードに関する情報が表示されます。ノードには 3 つのレベルがあります。

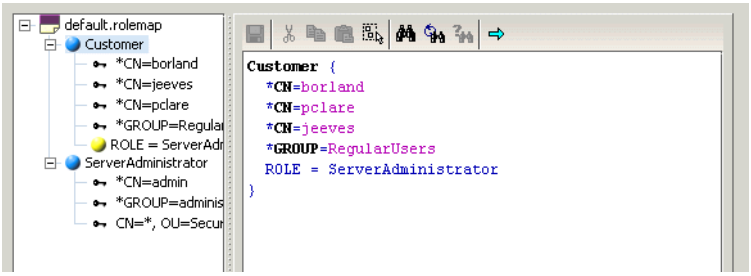
- **ドメイン：**このノードレベルには，承認ドメインに対応するロールマップファイルの名前が表示されます。子ノードはロールです。このノードを選択すると，ロールマップファイル全体が表示されます。

図 5.9　ドメインノード



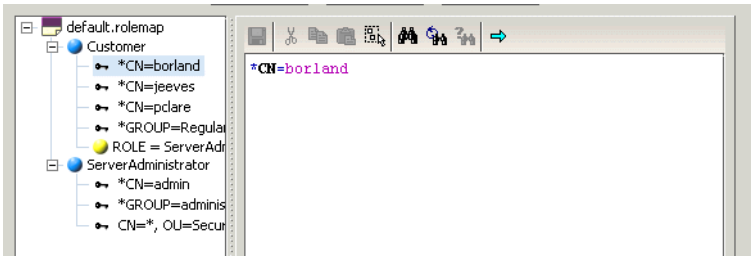
- **ロール：**このノードは，ドメインの特定のロールを表します。このノードを選択すると，ロール DB のロールエントリが表示されます。子ノードはルールです。

図 5.10　ロールノード



- **ルール：**このノードは，親ロールのアクセスルールを表します。このノードを選択すると，対応するロールエントリのルールエントリが表示されます。

図 5.11　ルールノード



承認ロールマップとドメインの使用

承認ロールマップを編集するには，次の手順にしたがいます。

1 ［Authorization Settings］ダイアログを開きます。

2 左側のペインでドメインノードを選択します。承認ロールマップファイルが表示されます。

3 内容ウィンドウでファイルを編集します。

4 終了したら，［OK］をクリックします。

新しい承認ロールマップを追加するには，次の手順にしたがいます。

1 ［Authorization Settings］ダイアログを開きます。

2 ［Add］ボタンをクリックします。［Add Domain］ダイアログボックスが表示されます。

3 新しいロールマップが属する新しいドメインの名前を入力します。

4 新しいロールマップが［Authorization Settings］ダイアログに表示されます。ウィンドウの上部にあるドロップダウンリストを使用すれば，ほかのロールマップに切り替えることができます。

5 必要に応じて，ロールとルールを追加します。

6 終了したら，［OK］をクリックします。

既存のロールマップのクローンを作成するには，次の手順にしたがいます。

1 ［Authorization Settings］ダイアログを開きます。

2 ［Clone］をクリックします。［Clone Domain］ダイアログボックスが表示されます。

3 クローンとして作成されるロールマップが属する新しいドメインの名前を入力します。

4 新しいロールマップが［Authorization Settings］ダイアログに表示されます。ウィンドウの上部にあるドロップダウンリストを使用すれば，ほかのロールマップに切り替えることができます。

5 必要に応じて，ロールとルールを追加または編集します。

6 終了したら，［OK］をクリックします。

承認ドメインとそのロールマップを削除するには，次の手順にしたがいます。

1 ［Authorization Settings］ダイアログを開きます。

2 ウィンドウの上部にあるドロップダウンボックスから削除するドメインを選択します。

3 ［Remove］ボタンをクリックします。ドメインが削除されます。

4 終了したら，［OK］をクリックします。

個々のロールの編集

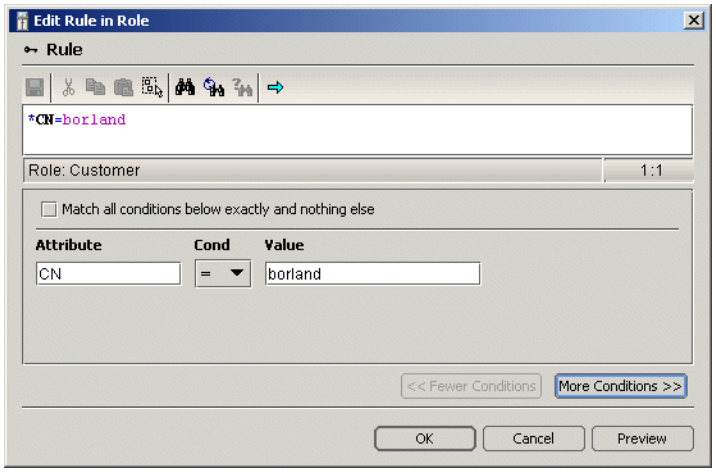
既存の承認ロールのルールを編集するには，次の手順にしたがいます。

1 ［Authorization Settings］ダイアログを開きます。

2 左側のペインでルールノードを選択します。内容ペインにルール表現が表示されます。

3 ［Edit］ボタンをクリックします。［Edit Rule in Role］ダイアログが表示されます。

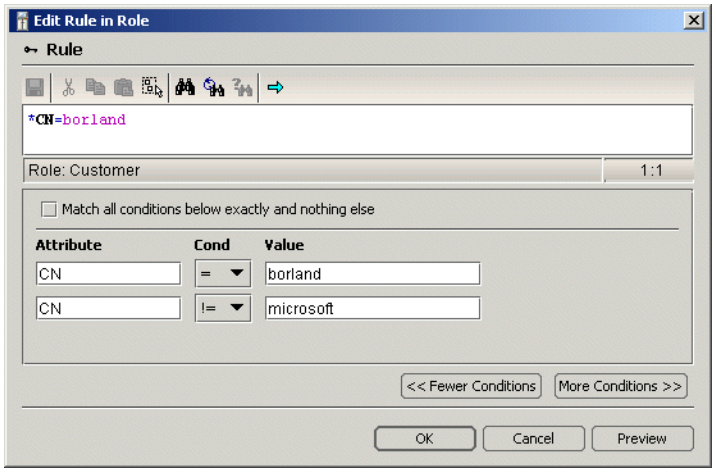
図 5.12 個々のロールの編集



4 表示されるボックスで属性，条件演算子，およびルールの値を編集します。

5 ルールに条件を追加する場合は，［More Conditions］をクリックします。新しい行が表示されます。

図 5.13 ［ロールのルール編集］ダイアログボックス



6 必要に応じて，属性，演算子，および値を追加します。最後の条件を削除する場合は，［Fewer Conditions］ボタンをクリックします。

7 ルールに基づいて厳密なアクセスを適用する場合は，［Match all conditions below exactly and nothing else］チェックボックスをチェックします。

8 編集したルールを表示するには，［preview］をクリックします。

9 終了したら，［OK］をクリックします。

ロールの追加および削除

新しいロールをロールマップに追加するには，次の手順にしたがいます。

1 ［Authorization Settings］ダイアログを開きます。

2 ロールを追加するドメインノードを右クリックします。コンテキストメニューから［New Role］を選択します。［New Role］ダイアログボックスが表示されます。

3 新しいロールに承認ドメイン内で一意の名前を付けて，［OK］をクリックします。

4 新しいロールがナビゲーションペインに表示されます。終了したら，［OK］をクリックします。

ロールマップから既存のロールを削除するには，次の手順にしたがいます。

1 ［Authorization Settings］ダイアログを開きます。

2 削除するロールノードを右クリックします。コンテキストメニューから［Delete Role］を選択します。

3 ロールマップファイルからロールが削除されます。終了したら，［OK］をクリックします。

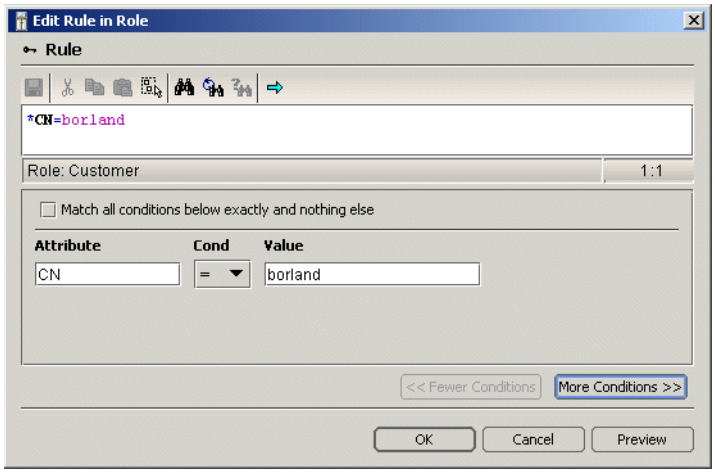
ルールの追加および削除

ルールをロールエントリに追加するには，次の手順にしたがいます。

1 ［Authorization Settings］ダイアログを開きます。

2 ルールを追加するロールノードを右クリックします。コンテキストメニューから［New Rule］を選択します。［New Rule in Role］ダイアログが表示されます。

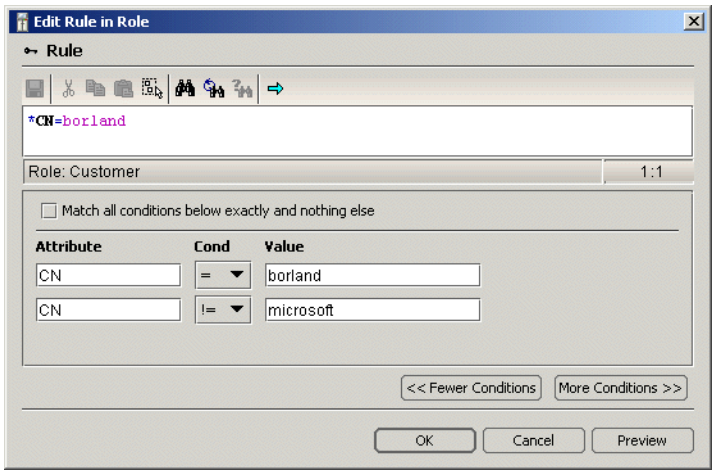
図 5.14 ルールの編集



3 表示されるボックスで属性，条件演算子，およびルールの値を編集します。

4 ルールに条件を追加する場合は，［More Conditions］をクリックします。新しい行が表示されます。

図 5.15　ルールへの条件の追加



5 必要に応じて，属性，演算子，および値を追加します。最後の条件を削除する場合は，［Fewer Conditions］ボタンをクリックします。

6 ルールに基づいて厳密なアクセスを適用する場合は，［Match all conditions below exactly and nothing else］チェックボックスをチェックします。

7 編集したルールを表示するには，［preview］をクリックします。

8 終了したら，［OK］をクリックします。新しいルールがツリーに表示されます。

9 終了したら，［OK］をクリックします。

ロールからルールを削除するには，次の手順にしたがいます。

1 ［Authorization Settings］ダイアログを開きます。

2 削除するルールを右クリックします。

3 コンテキストメニューから［Delete Rule］を選択します。ルールがツリーから削除されます。

4 終了したら，［OK］をクリックします。

VisiSecure プロパティの指定

各プロファイルには，VisiSecure の動作をカスタマイズするための security.properties ファイルが入っています。次に，典型的なプロパティファイルの例を示します。

```
# デフォルトでユーザードメインのセキュリティを無効化する
vbroker.security.disable=false

# ORB の認証設定ファイルをポイントする
vbroker.security.login=false
vbroker.security.authentication.config=${profile.path}/config.jaas

# 提供された承認ドメインに名前を付ける
vbroker.security.authDomains=default
vbroker.security.authDomains.default=default

# ロールマップファイルにないメソッドの要求の処理方法（grant または deny）
vbroker.security.domain.default.defaultAccessRule=grant
vbroker.security.domain.default.rolemap_path=${profile.path}/default.rolemap
```

アプリケーションは Java，C++，またはその両方を使用している場合があり，それぞれの場合によって異なるプロパティに異なる種類の値を設定する必要があります。このファイルに設定できるすべてのプロパティについては，第 10 章「セキュリティプロパティ（C++）」と第 9 章「セキュリティプロパティ（Java）」を参照してください。

すべてのプロパティを設定すると，security.properties ファイルがプロファイルのフォルダに保存されます。

プロファイルとドメインの関連付け

セキュリティプロファイルは，プロパティを設定することで，システムのさまざまなドメインに関連付けることができます。プロパティを設定すると，VisiSecure はこの設定を使って関連付けられているセキュリティプロファイルを探し，ドメインのセキュリティを保護します。システムの各ドメインには，orb.properties ファイルが関連付けられています。この設定ファイルは，次の場所にあります。

```
<install-dir>/var/domains/<domain_name>/adm/properties/orb.properties
```

プロファイルおよびその設定をドメインに関連付けるには

1. ドメインの orb.properties ファイルを開きます。
2. 設定内容をドメインに使用するプロファイルの名前を vbroker.security.profile プロパティに設定します。次に例を示します。

```
vbroker.security.profile=default
```

これで，VisiSecure はこのドメインに対してセキュリティ操作を実行する際に，指定されたセキュリティプロファイルの設定を参照します。

ドメインに対するボールトの使用

ボールトを使ってシステム ID を保存する場合は，ボールトを使用できるようにドメインに関連付けます。それには，ドメインの orb.properties ファイルの vbroker.security.vault プロパティを設定します。このプロパティにドメインのボールトの場所を設定します。次に例を示します。

```
vbroker.security.vault=c:/BDP/var/domains/base/adm/security/MyVault
```

orb.properties ファイルだけに属しているほかのプロパティもボールトと同様です。これには，セキュリティで保護されているリスナーポート，スレッドの監視などがあります。一般に，複数のアプリケーションで共有できるプロファイルにはこのプロパティだけを追加します。それ以外のプロパティは，適切なプロセス固有の ORB プロパティファイルを使って指定します。

第 6 章

セキュリティで保護された接続の作成（Java）

ここでは，VisiSecure を使って Java アプリケーション用のセキュリティで保護された接続を作成する方法について説明します。JSSE（Java Secure Sockets Extension）に関して簡単に説明した後で，アプリケーションのセキュリティ保護に関する詳細な手順についても説明します。

JAAS と JSSE

VisiSecure では，J2EE アプリケーションでクライアントとサーバーとの相互認証を行うために，JAAS（Java Authentication and Authorization Service）を使用します。JAAS は，ユーザー認証と権限の割り当てに使用するフレームワークと標準インターフェースを提供します。Borland VisiBroker Server は，JSSE（Java Secure Socket Extension）を使って SSL をサポートするためのメカニズムを提供します。

JAAS がサービスの提供に使用する用語については，認証に関する第 3 章「認証」参照してください。

JAAS の基本概念

VisiBroker ORB は，通信プロトコルとして IIOP（Internet Inter-ORB Protocol）を使用します。JSSE（Java Secure Sockets Extension）は，セキュリティで保護されたインターネット接続を有効にします。これは SSL と TLS プロトコルの Java インプリメンテーションで，データの暗号化機能，サーバーとクライアントの認証，およびメッセージの整合性があります。また，JSSE は Java アプリケーションで簡単に直接実装できる基本要素の役割を果たします。

JSSE は API だけでなく，API のインプリメンテーションも提供しています。インプリメンテーションには，ソケットクラス，トラストマネージャ，キーマネージャ，SSLContexts，ソケットファクトリフレームワーク，および公開キー証明書 API があります。

JSSE は，SSL インプリメンテーションの一部である基底のハンドシェイクメカニズムをサポートしています。この SSL インプリメンテーションには，暗号化スイートのネゴシエーション，クライアント／サーバー認証，サーバーのセッション管理，および RSA Data Security, Inc. のライセンスコードがあります。JSSE は Java KeyStores を証明書と秘密

キーのリポジトリとして使用します。KeyStores の詳細については，Sun Microsystems の JDK ドキュメントを参照してください。JSSE プロパティを使用して，信頼できる KeyStore と ID KeyStore を指定できます。

セキュリティで保護されたクライアントとサーバーの設定

次は，セキュリティで保護されたクライアントやサーバーを開発するために必要な操作で，クライアントとサーバーで共通しているものです。CORBA ユーザーのプロパティは，設定ファイルを通して配置されるファイルにすべて保存されます。クライアントとサーバーの使用モデルを個別に説明します。ナビゲーションペインで目的のノードを右クリックして［Edit Properties］を選択すると，すべてのプロパティを VisiBroker 管理コンソールで設定できます。

メモ これらの手順は，Java と C++ アプリケーションでほとんど同じです。

重要 RoleDB，LoginModule 設定などのすべてのセキュリティ情報は，管理コンソールの該当するプロパティタブで設定できます。

ステップ 1 ：ID の指定

ID は，ユーザー名，パスワード，および領域の 3 つの情報を組み合わせるか，または証明書を使用することもできます。これらは JAAS モジュールまたは API を使って収集します。

クライアント ユーザー名とパスワードを使用するクライアントでは，サーバーの領域に関してクライアントが認識できる内容には一定の制限があります。ID を照会した時点でクライアントは，サーバーがサポートしている領域に関する詳しい情報を持っている場合も，反対にまったく情報がない場合もあります。また，クライアントはサーバーエンドで認証されるので，注意してください。

サーバー ユーザー名とパスワードという ID を使用しているサーバーの場合，領域は常に知られているため，認証はローカルで実行されます。

証明書 ID も，KeyStore に格納されているのか，API を介して指定されたかによって，制限がある場合もあります。

こうした制限に注意して，Borland VisiBroker Server では，サーバーまたはクライアントへの ID の提示に使用する次のようなモデルをサポートしています。

- 既知の領域に対して，JAAS モジュールを使用したユーザー名／パスワードの認証
- API を使用したユーザー名／パスワードの認証
- プロパティ設定を通して KeyStore を使用した証明書ベースの認証
- API を通して KeyStore を使用した証明書ベースの認証
- API を使用した証明書ベースの認証
- KeyStore を使用した pkcs12 ベースの認証
- API を使用した pkcs12 ベースの認証

既知の領域に対して，JAAS モジュールを使用したユーザー名／パスワードの認証

クライアントが認証しようとする領域が既知の場合，クライアント側の JAAS 設定は次のような形式を使用します。

```
vbroker.security.login=true
vbroker.security.login.realms=<known-realm>
```

API を使用したユーザー名／パスワードの認証

次のサンプルコードでは，ログイン API の使用をシミュレートしています。ここれは，wallet を使用しています。サポートされている 4 つのログインモードの詳細については，VisiSecure for Java API および「VisiSecure SPI for Java」を参照してください。

```
public static void main(String[] args) {
   //ORB の初期化
   org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
   com.borland.security.Context ctx = (com.borland.security.Context)
      orb.resolve_initial_references("VBSecurityContext");
   if(ctx != null) {
      com.borland.securty.IdentityWallet wallet =
         new com.borland.security.IdentityWallet(<username>,
            <password>.toCharArray(), <realm>);
      ctx.login(wallet);
   }
}
```

プロパティ設定を通して KeyStore を使用した証明書ベースの認証

vbroker.security.login.realms=Certificate#ALL プロパティを設定すると，クライアントはキーストアの場所とアクセス情報の入力を求められます。有効値については，認証に関する第 3 章「認証」を参照してください。

API を通して KeyStore を使用した証明書ベースの認証

KeyStores による証明書を使ってログインするには，「API を使用したユーザー名／パスワードの認証」で説明した API と同じものを使用します。IdentityWallet にある領域名は，CERTIFICATE#ALL にしてください。username は，デフォルトの KeyStore にあって Key エントリを参照しているエリアスです。password は，同じ Key エントリに対応している秘密キーのパスワード（KeyStore パスワードと同じ）です。

API を使用した証明書ベースの認証

KeyStore を直接使用しない場合は，CertificateWalletAPI を使って証明書と秘密キーを指定します。このクラスは，pkcs12 ファイル形式もサポートしています。

```
X509Certificate[] certChain = ...list-of-X509-certificates...
PrivateKey privKey = private-key
com.borland.security.CertificateWallet wallet =
   new com.borland.security.CertificateWallet(alias,
      certChain, privKey, "password".toCharArray());
```

新しい CertificateWallet に最初の引数があれば，この引数は KeyStore のエントリのエリアスです。KeyStore を使用しない場合は，引数を null に設定してください。

KeyStore を使用した pkcs12 ベースの認証

pkcs12 KeyStores を使ってログインするには，61 ページの「API を使用したユーザー名／パスワードの認証」で説明した API と同じものを使用します。IdentityWallet にある領域名は，CERTIFICATE#ALL にしてください。username は，デフォルトの KeyStore にあって Key エントリを参照しているエリアスで，password は，pkcs 12 ファイルのロック解除に必要なパスワードです。javax.net.ssl.KeyStore プロパティには，データベースの場所を指定します。

API を使用した pkcs12 ベースの認証

61 ページの「API を使用した証明書ベースの認証」を参照してください。

ステップ 2：プロパティと保護品質（QoP）の設定

接続の保護品質を確認するため，いくつかのプロパティを使用します。接続の品質を調整する場合は，第 9 章「セキュリティプロパティ（Java）」を参照してください。たとえば，暗号の強度を設定する場合は，SSL 接続の cipherList プロパティを設定します。

サーバーには ServerQoPConfig API，クライアントには ClientQoPConfig API を使用して，QoP ポリシーを設定します。API を使用すると，ターゲットトラスト（ターゲットを認証するかどうか），それにトランスポートポリシー（SSL や別に指定したセキュリティで保護されているトランスポートメカニズムを使用するかどうか）を設定することができ，またサーバーの場合には，POA オブジェクトにアクセスポリシーを設定する RoleDB にアクセスする AccessPolicyManager も設定できます。QoP API の詳細については，VisiSecure for Java API および SPI のマニュアルを参照してください。

ステップ 3：信頼の設定

X509TrustManager インターフェースインプリメンテーションを提供するには，該当するセキュリティコンテキストに API setTrustManager を使用してください。認証を受ける必要がある証明書は KeyStore に置き，javax.net.ssl.trustStore プロパティを使用してこの証明書の設定を行います。トラストマネージャがない場合は，セキュリティサービスが提供するデフォルトの X509TrustManager が使用されます。

ほかのトラストポリシーは QoP 設定で設定します。詳細については，62 ページの「ステップ 2：プロパティと保護品質（QoP）の設定」を参照してください。

ステップ 4：擬似ランダム番号（Pseudo-Random Number：PRNG）ジェネレータの設定

SSL 通信を使用する場合は，PRNG を設定する必要があります。SecureRandom オブジェクトを作成して送信します。com.borland.security.Context インターフェースの setSecureRandom メソッドを使用してこのオブジェクトを PRNG として設定します。com.borland.security.Context インターフェースの詳細については，VisiSecure for Java API および SPI のマニュアルを参照してください。

ステップ 5：必要な場合の ID アサーションの設定

クライアントが中間層サーバーのメソッドを呼び出すと，この要求のコンテキストの範囲内で，最終層サーバーが呼び出されます。デフォルトでは，クライアントの ID は，中間層サーバーによって内部でアサーションが行われます。このため，最終層サーバーで getCallerPrincipal を呼び出すと，クライアントのプリンシパルが返されます。これで，クライアントの ID は中間層サーバーアサーションが行われます。ID はユーザー名と証明書のどちらでもかまいません。秘密キーやパスワードのようなクライアントの認証情報は，アサーションに送られることはありません。これは，このような ID は最終層で認証できないことを意味します。

ユーザーがデフォルトの ID アサーションを上書きする場合は，特定のプリンシパルのアサーションに利用できる API があります。これらの API は，呼び出しのコンテキスト内にある中間層サーバー上で，特別な許可がある場合にだけ呼び出されます。詳細については，VisiSecure for Java API および SPI のマニュアルを参照してください。

SSL 関連情報のチェック

Borland VisiBroker Server では，SSL の関連情報を詳しく調べて設定する API を提供しています。SecureContext API は，Trust Manager や PRNG の指定，SSL 暗号化スイートの調査，および選択した暗号の有効化に使用します。

クライアント ピアの証明書を調べるには，getPeerSession() を使用して，ターゲットに関連付けられた SSLSession オブジェクトを返します。これで，情報を取得する標準 JSSE API を使用することができます。

サーバー サーバー側のピア証明書を調べるには，com.borland.security.Context を使って SSL 接続を設定し，com.borland.security.Current と API を使ってスレッドに関連付けられている SSLSession を調べます。

カスタムプラグインの作成

VisiSecure にはカスタムプラグインを使用できるさまざまなコンポーネントがあります。

- LoginModules
- コールバックハンドラ
- SPI 経由の承認サービスプロバイダ
- SPI 経由のアサーションの信頼

LoginModules

javax.security.auth.spi.LoginModules を拡張して，独自の LoginModule を実装できます。LoginModule を使用するには，ほかの LoginModule と同様に，認証設定ファイルで設定する必要があります。新しくカスタマイズされたモジュールは，セキュリティで保護されたアプリケーションによって実行時にロードする必要があります。

認証設定の構文は次のとおりです。

```
<realm-name> {
    <class-name-of-customized-LoginModule> <required|optional>;
}
```

CallbackHandlers

javax.security.auth.callback.CallBackHandler を拡張して，独自のコールバックを実装できます。コールバックを使用するには，ほかのコールバックハンドラと同様に，セキュリティプロパティファイルの vbroker.security.authentication.callbackHandler=<custom-handler-class-name> プロパティを設定する必要があります。新しくカスタマイズされたモジュールは，セキュリティで保護されたアプリケーションによって実行時にロードする必要があります。

承認サービスプロバイダ

承認は，特定のリソースのためにセキュリティ属性または権限に基づいてアクセスコントロールを決定するプロセスです。VisiSeucre は，承認においてアクセス許可の概念を使用します。RolePermission クラスは，「ロール」をアクセス許可として表すために定義します。次に，承認サービスプロバイダは，権限を特定のリソースに関連付けるロールアクセス許可の同種コレクションのインプリメンテーションを提供します。

承認サービスプロバイダは，承認ドメインと密接に対応しています。各ドメインに実装される承認サービスプロバイダは 1 つです。ORB（Object Request Broker）の初期化中に vbroker.security.authDomains によって定義された承認ドメインが構築され，その間に承認サービスプロバイダインプリメンテーションのインスタンスが作成されます。

承認サービスをプラグインするには，次のプロパティを設定する必要があります。

```
vbroker.security.auth.domains=MyDomain
vbroker.security.domain.MyDomain.provider=MyProvider
vbroker.security.domain.MyDomain.property1=xxx
vbroker.security.domain.MyDomain.property2=xxx

vbroker.security.identity.attributeCodecs=MyCodec
vbroker.security.adapter.MyCodec.property1=xxx
vbroker.security.adapter.MyCodec.property2=xxx
```

指定するプロパティは，上と同じメカニズムにしたがってユーザープラグインに渡されます。

信頼プロバイダ

アサーション信頼メカニズムをプラグインすることもできます。アサーションは複数のホップシナリオで発生するか，またはアサーション API を介して明示的に呼び出されます。サーバーには，ピアが信頼されてアサーションを生成できるかどうかを決定するルールを設定できます。デフォルトのインプリメンテーションは，プロパティ設定を使ってサーバー側で信頼されたピアを設定します。ピアは実行時に認証と承認を渡し，アサーションを生成するための信頼を確立する必要があります。信頼プロバイダは，セキュリティサービス全体で 1 つだけです。

アサーション信頼メカニズムをプラグインするには，次のプロパティを設定する必要があります。

```
vbroker.security.trust.trustProvider=MyProvider
vbroker.security.trust.trustProvider.MyProvider.property1=xxx
vbroker.security.trust.trustProvider.MyProvider.property2=xxx
```

指定するプロパティは，上と同じメカニズムにしたがってユーザープラグインに渡されます。

第 7 章

セキュリティで保護された接続の作成（C++）

ここでは，VisiSecure を使って C++ アプリケーション用のセキュリティで保護された接続を作成する方法について説明します。

セキュリティで保護されたクライアントとサーバーの設定

次は，セキュリティで保護されたクライアントやサーバーを開発するために必要な操作で，クライアントとサーバーで共通しているものです。CORBA ユーザーのプロパティは，設定ファイルを通して配置されるファイルにすべて保存されます。クライアントとサーバーの使用モデルを個別に説明します。ナビゲーションペインで目的のノードを右クリックして［Edit Properties］を選択すると，すべてのプロパティを BES 管理コンソールで設定できます。

メモ これらの手順は，Java と C++ アプリケーションでほとんど同じです。

重要 RoleDB，LoginModule 設定などのすべてのセキュリティ情報は，管理コンソールの該当するプロパティタブで設定できます。

ステップ 1 ：ID の指定

ユーザー名，パスワード，領域という 3 つの情報を組み合わせて ID にしたり，証明書を使用することができます。これらは LoginModule または API を使って収集されます。

クライアント ユーザー名とパスワードを使用するクライアントでは，サーバーの領域に関してクライアントが認識できる内容には一定の制限があります。ID を照会した時点でクライアントは，サーバーがサポートしている領域に関する詳しい情報を持っている場合も，反対にまったく情報がない場合もあります。また，クライアントはサーバーエンドで認証されるので，注意してください。

サーバー ユーザー名とパスワードという ID を使用しているサーバーの場合，領域は常に知られているため，認証はローカルで実行されます。

証明書 ID も，KeyStore に格納されているのか，API を介して指定されたかによって，制限がある場合もあります。VisiSecure for C++ の KeyStore は，証明書チェインが入っている `trustpointRepository` と同様のディレクトリ構造を参照します。

こうした制限に注意して，Borland VisiBroker では，サーバーまたはクライアントへの ID の提示に使用する次のようなモデルをサポートしています。

- 既知の領域に対して，LoginModule を使用したユーザー名／パスワードの認証
- API を使用したユーザー名／パスワードの認証
- プロパティ設定を通して KeyStore を使用した証明書ベースの認証
- API を通して KeyStore を使用した証明書ベースの認証
- API を使用した証明書ベースの認証
- KeyStore を使用した pkcs12 ベースの認証
- API を使用した pkcs12 ベースの認証

既知の領域に対して，LoginModule を使用したユーザー名／パスワードの認証

クライアントが認証しようとする領域が既知の場合，クライアント側の設定は次のような形式を使用します。

```
vbroker.security.login=true
vbroker.security.login.realms=<known-realm>
```

API を使用したユーザー名／パスワードの認証

次のサンプルコードでは，ログイン API の使用をシミュレートしています。ここれは，wallet を使用しています。サポートされている 4 つのログインモードの詳細については，第 11 章「VisiSecure for C++ API」および第 12 章「Security SPI for C++」を参照してください。

```
int main(int argc, char* const* argv) {
    // ORB の初期化
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
    CORBA::Object_var obj = orb-
>resolve_initial_references("VBSecurityContext");
    Context* c = dynamic_cast<Context*> (obj.in());
    // walletFactory の取得
    CORBA::Object_var o = orb->resolve_initial_references("VBWalletFactory");
    vbsec::WalletFactory* wf = dynamic_cast<vbsec::WalletFactory*>(o.in());
    vbsec::Wallet* wallet = wf->createIdentityWallet( <username>, <password>,
<realm>);
    c->login(*wallet);
}
```

プロパティ設定を通して KeyStore を使用した証明書ベースの認証

vbroker.security.login.realms=Certificate#ALL プロパティを設定すると，クライアントはキーストアの場所とアクセス情報の入力を求められます。有効値については，認証に関する第 3 章「認証」を参照してください。

API を通して KeyStore を使用した証明書ベースの認証

KeyStores による証明書を使ってログインするには，66 ページの「API を使用したユーザー名／パスワードの認証」で説明した API と同じものを使用します。IdentityWallet にある領域名は，CERTIFICATE#ALL にしてください。username は，デフォルトの KeyStore にあって Key エントリを参照しているエリアスです。password は，同じ Key エントリに対応している秘密キーのパスワード（KeyStore パスワードと同じ）です。

API を使用した証明書ベースの認証

KeyStore を直接使用しない場合は，CertificateFactoryAPI を使って証明書と秘密キーをインポートします。このクラスは，pkcs12 ファイル形式もサポートしています。

```
CORBA::Object_var o = orb-
>resolve_initial_references("VBSecureSocketProvider");
vbsec::SecureSocketProvider* ssp =
dynamic_cast<vbsec::SecureSocketProvider*>(o.in());

const vbsec::CertificateFactory& cf = ssp->getCertificateFactory ();
```

新しい CertificateWallet に最初の引数があれば，この引数は KeyStore のエントリのエリアスです。KeyStore を使用しない場合は，引数を null に設定してください。

KeyStore を使用した pkcs12 ベースの認証

pkcs12 KeyStores を使ってログインするには，66 ページの「API を使用したユーザー名／パスワードの認証」で説明した API と同じものを使用します。IdentityWallet にある領域名は，CERTIFICATE#ALL にしてください。username は，デフォルトの KeyStore にあって Key エントリを参照しているエリアスで，password は，pkcs 12 ファイルのロック解除に必要なパスワードです。javax.net.ssl.KeyStore プロパティには，データベースの場所を指定します。

API を使用した pkcs12 ベースの認証

66 ページの「API を使用した証明書ベースの認証」を参照してください。

ステップ 2：プロパティと保護品質（QoP）の設定

接続の保護品質を確認するため，いくつかのプロパティを使用します。接続の品質を調整する場合は，第 10 章「セキュリティプロパティ（C++）」を参照してください。たとえば，暗号の強度を設定する場合は，SSL 接続の cipherList プロパティを設定します。

サーバーには ServerQoPConfig API，クライアントには ClientQoPConfig API を使用して，QoP ポリシーを設定します。API を使用すると，ターゲットトラスト（ターゲットを認証するかどうか），それにトランスポートポリシー（SSL や別に指定したセキュリティで保護されているトランスポートメカニズムを使用するかどうか）を設定することができ，またサーバーの場合には，POA オブジェクトにアクセスポリシーを設定する RoleDB にアクセスする AccessPolicyManager も設定できます。

ステップ 3：信頼の設定

信頼は，vbroker.security.trustpointRepository=Directory:<path to directory> プロパティを使って設定します。このディレクトリには信頼された証明書が保存されます。

ほかのトラストポリシーは QoP 設定で設定します。詳細については，67 ページの「ステップ 2：プロパティと保護品質（QoP）の設定」を参照してください。

ステップ 4：必要な場合の ID アサーションの設定

クライアントが中間層サーバーのメソッドを呼び出すと，この要求のコンテキストの範囲内で，最終層サーバーが呼び出されます。デフォルトでは，クライアントの ID は，中間層サーバーによって内部でアサーションが行われます。このため，最終層サーバーで getCallerSubject を呼び出すと，クライアントのプリンシパルが返されます。これで，クライアントの ID は中間層サーバーアサーションが行われます。ID はユーザー名と証明書のどちらでもかまいません。秘密キーやパスワードのようなクライアントの認証情報は，アサーションに送られることはありません。これは，このような ID は最終層で認証できないことを意味します。

ユーザーがデフォルトの ID アサーションを上書きする場合は，特定のプリンシパルのアサーションに利用できる API があります。これらの API は，呼び出しのコンテキスト内にある中間層サーバー上で，特別な許可がある場合にだけ呼び出されます。

SSL 関連情報のチェック

Borland VisiBroker では，SSL の関連情報を詳しく調べて設定する API を提供しています。SecureContext API は，SSL 暗号化スイートの調査および選択した暗号の有効化に使用します。

クライアント ピアの証明書を調べるには，getPeerSession() を使用して，ターゲットに関連付けられた SSLSession オブジェクトを返します。これで，情報を取得する標準 JSSE API を使用することができます。

サーバー サーバー側のピア証明書を調べるには，com.borland.security.Context を使って SSL 接続を設定し，com.borland.security.Current と API を使ってスレッドに関連付けられている SSLSession を調べます。

カスタムプラグインの作成

VisiSecure にはカスタムプラグインを使用できるさまざまなコンポーネントがあります。

- LoginModules
- コールバックハンドラ
- SPI 経由の承認サービスプロバイダ
- SPI 経由のアサーションの信頼

VisiSecure for C++ がユーザーのインプリメンテーションを検索できるように，すべてのプラグインは，VisiSecure が提供する REGISTER_CLASS マクロを使ってセキュリティサービスにクラスを登録する必要があります。登録されたクラスを指定する場合は，名前空間とともに完全なクラス名を指定する必要があります。名前空間は，最も外側の名前空間から始まり，「.」または「::」で区切られた文字列という標準形式で指定する必要があります。次に例を示します。

```
MyNameSpace {
  class MyLoginModule {
    ......
  }
}
```

これは，MyNameSpace.MyLoginModule または MyNameSpace::MyLoginModule と指定します。

LoginModules

vbsec::LoginModule を拡張して，独自の LoginModule を実装できます。LoginModule を使用するには，ほかの LoginModule と同様に，認証設定ファイルで設定する必要があります。新しくカスタマイズされたモジュールは，セキュリティで保護されたアプリケーションによって実行時にロードする必要があります。

認証設定の構文は次のとおりです。

```
<realm-name> {
    <class-name-of-customized-LoginModule> <required|optional>;
}
```

メモ VisiSecure が暗黙的に「.」を「::」に置換します。
したがって，com.borland.security.provider.authn.HostLoginModule は，com::borland::security::provider::authn::HostLoginModule と同じです。

CallbackHandlers

vbsec::CallbackHandler を拡張して，独自のコールバックを実装できます。コールバックを使用するには，ほかのコールバックハンドラと同様に，セキュリティプロパティファイルの vbroker.security.authentication.callbackHandler=<custom-handler-class-name> プロパティを設定する必要があります。新しくカスタマイズされたモジュールは，セキュリティで保護されたアプリケーションによって実行時にロードする必要があります。

承認サービスプロバイダ

承認は，特定のリソースのためにセキュリティ属性または権限に基づいてアクセスコントロールを決定するプロセスです。VisiSeucre は，承認においてアクセス許可の概念を使用します。RolePermission クラスは，「ロール」をアクセス許可として表すために定義します。次に，承認サービスプロバイダは，権限を特定のリソースに関連付けるロールアクセス許可の同種コレクションのインプリメンテーションを提供します。

承認サービスプロバイダは，承認ドメインと密接に対応しています。各ドメインに実装される承認サービスプロバイダは 1 つです。ORB（Object Request Broker）の初期化中に vbroker.security.authDomains によって定義された承認ドメインが構築され，その間に承認サービスプロバイダインプリメンテーションのインスタンスが作成されます。

承認サービスをプラグインするには，次のプロパティを設定する必要があります。

```
vbroker.security.auth.domains=MyDomain
vbroker.security.domain.MyDomain.provider=MyProvider
vbroker.security.domain.MyDomain.property1=xxx
vbroker.security.domain.MyDomain.property2=xxx

vbroker.security.identity.attributeCodecs=MyCodec
vbroker.security.adapter.MyCodec.property1=xxx
vbroker.security.adapter.MyCodec.property2=xxx
```

指定するプロパティは，上と同じメカニズムにしたがってユーザープラグインに渡されます。

信頼プロバイダ

アサーション信頼メカニズムをプラグインすることもできます。アサーションは複数のホップシナリオで発生するか，またはアサーション API を介して明示的に呼び出されます。サーバーには，ピアが信頼されてアサーションを生成できるかどうかを決定するルールを設定できます。デフォルトのインプリメンテーションは，プロパティ設定を使ってサーバー側で信頼されたピアを設定します。ピアは実行時に認証と承認を渡し，アサーションを生成するための信頼を確立する必要があります。信頼プロバイダは，セキュリティサービス全体で 1 つだけです。

アサーション信頼メカニズムをプラグインするには，次のプロパティを設定する必要があります。

```
vbroker.security.trust.trustProvider=MyProvider
vbroker.security.trust.trustProvider.MyProvider.property1=xxx
vbroker.security.trust.trustProvider.MyProvider.property2=xxx
```

指定するプロパティは，上と同じメカニズムにしたがってユーザープラグインに渡されます。

第 8 章

Web コンポーネントのセキュリティ

Borland VisiBroker Server では，暗号化，認証，および承認の規則を使用して，Web コンポーネントをセキュリティで保護することができます。

J2EE 規格によって提供されているセキュリティ対策と同様に，セキュリティはモジュールレベルで設定します。また設定ファイル内でセキュリティメカニズムを宣言することで，Web コンポーネントをセキュリティで保護できます。

Apache Web Server のセキュリティ

Apache Web Server では，セキュリティのためにデータ転送の暗号化技術を使用しています。そのため，VisiBroker は，mod_ssl モジュールをサポートしています。このモジュールは，Open Source toolkit for SSL/TLS の OpenSSL を利用することで，Secure Sockets Layer（SSL v2/v3）および Transport Layer Security（TLS v1）プロトコルを実装した，強固な暗号化技術を Apache Web Server に提供します。

VisiBroker Server は，mod_ssl を提供します。OpenSSL バージョン 0.9.6g 準拠の mod_ssl は，今では Apache Web サーバーも直接サポートしています。

mod_ssl に応じた Apache 設定ファイルの変更

mod_ssl を有効にするには，<install_dir>/var/domains/<domain_name>/configurations/<configuration_name>/mos/<apache_ManagedObject_name>/conf にある httpd.conf ファイルの次の行のコメントを解除する必要があります。

```
LoadModule ssl_module <install_dir>/lib/<apache_ManagedObject_name>/mod_ssl.so
```

mod_ssl 指示文は，次のとおりです。

```
<IfModule mod_ssl.c>
SSLEngine on
SSLRandomSeed startup builtin
SSLRandomSeed connect builtin
SSLCertificateFile <install_dir>/var/domains/<domain_name>/configurations/
<configuration_name>/mos/<apache_ManagedObject_name>/conf/ssl.crt/server.crt
SSLCertificateKeyFile <install_dir>/var/domains/<domain_name>/configurations/
<configuration_name>/mos/<apache_ManagedObject_name>/conf/ssl.key/server.key
#
# 次のコメントをはずすと，SSL 証明書と関連情報が有効になり，Apache 環境にエキ
スポートされる

#SSLOptions +StdEnvVars +ExportCertData

</IfModule>
```

警告 KeepAlive 接続オプションをオン（デフォルト）にして mod-ssl Apache モジュールを使用すると，Apache Web サーバーが不安定になります。これは既知の問題です。mod-ssl Apache モジュールを有効にした後は注意が必要です。

表 8.1　mod_ssl 指示文

SSL 固有の指示文	説明
SSLEngine	SSLEngine の配置は重要です。これらは，サーバーレベルと特定の仮想ホストのいずれにも配置できます。前者の場合，サーバーは HTTPS 接続だけに応答します。後者の場合，特定のポート番号（通常は 443）に関連付けることができ，これによって標準 HTTP 接続と HTTPS 接続の両方を処理できます。
SSLRandomSeed	mod_ssl 暗号機能が使用する不規則性のソースを指定します。mod_ssl に組み込まれた不規則性でも十分ですが，実用的なセキュリティ環境で使用するには不十分です。/dev/random，/dev/urandom などの UNIX 不規則デバイスを使用してください。SSLRandomSeed 指示文は，サーバーレベルで定義してください。
SSLCertificateFile	SSLCertificateFile の配置は重要です。これらは，サーバーレベルと特定の仮想ホストのいずれにも配置できます。前者の場合，サーバーは HTTPS 接続だけに応答します。後者の場合，特定のポート番号（通常は 443）に関連付けることができ，これによって標準 HTTP 接続と HTTPS 接続の両方を処理できます。このファイルには任意の名前を付けることができ，アクセスできるディレクトリであればどこにでも配置できます。
SSLCertificateKeyFile	SSLCertificateKeyFile の配置は重要です。これらは，サーバーレベルと特定の仮想ホストのいずれにも配置できます。前者の場合，サーバーは HTTPS 接続だけに応答します。後者の場合，特定のポート番号（通常は 443）に関連付けることができ，これによって標準 HTTP 接続と HTTPS 接続の両方を処理できます。このファイルには任意の名前を付けることができ，アクセスできるディレクトリであればどこにでも配置できます。
SSLOptions +StdEnvVars +ExportCertData	デフォルトではコメントアウトされています。74 ページの「Borland Web コンテナへの証明書パススルーの有効化」場合は，この指示文のコメントを解除し，mod_ssl が SSL 証明書およびその関連情報をブラウザを使って共有環境に渡すようにする必要があります。

重要 VisiBroker Server は key_file と certificate_file を提供していないので生成する必要があります。72 ページの「キーファイルと証明書ファイルの作成」を参照してください。

mod_ssl 設定の詳細については，以下のサイトを参照してください。

http://www.modssl.org/docs

キーファイルと証明書ファイルの作成

VisiBroker Server は，「openssl」ユーティリティを備えており，mod_ssl に必要なキーファイルや証明書ファイルを生成できます。openssl ユーティリティは次の場所にあります。

```
<install_dir>/bin/<apache_ManagedObject_name>/openssl/
```

- Windows の場合，openssl 実行可能プログラムをダブルクリックすると，コマンドウィンドウが表示されます。
- UNIX の場合，次の手順にしたがいます。

メモ　UNIX の場合は，シーディング用の不規則データソースも必要です。/dev/rand デバイスがインストールされていない場合，512 バイトより長い乱数でファイルを用意します。

openssl 実行可能プログラムにより，まず「RANDFILE」という名前の変数の環境が検索されます。検索する値は，少なくとも 512 バイトのデータを格納したファイルです。環境変数 RANDFILE が見つからない場合，実行可能プログラムはホームディレクトリのルートで，<file_name>.rnd を検索します。それが見つかると，シーディング用の最低 512 バイトのデータを格納したファイルであるとみなされます。/dev/rand デバイスがインストールされておらず，ほかの代替要素も提供されない場合，証明書の生成は失敗します。

ファイルを生成するには，次の手順にしたがいます。

1 サーバーの共有キーを作成します。

```
OpenSSL> genrsa -out <key_file>
```

2 認証要求を生成します。

```
OpenSSL> req -new -key <key_file> -out <request_file> -config <install_dir>/bin/
<apache_ManagedObject_name>/openssl.cnf
```

3 一時認証を作成します。

```
OpenSSL> req -x509 -key <key_file> -in <request_file> -out <certificate_file> -
config <install_dir>/bin/<apache_ManagedObject_name>/openssl.cnf
```

次の設定を使用します。

```
<install_dir>/bin/<apache_ManagedObject_name>/openssl.cnf
```

以下の情報が要求されます。

メモ　各照会に対して（デフォルト値を追認して）〔Enter〕を押すだけで一時認証を作成できます。

```
Using configuration from
<install_dir>/bin/<apache_ManagedObject_name>/openssl.cnf
You are about to be asked to enter information that will be incorporated into
your certificate request.
What you are about to enter is what is called a Distinguished Name or a
DN.
There are quite a few fields but you can leave some blank.
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (such as, city) []:
Organization Name (such as, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (such as, section) []:
Common Name (such as, YOUR name) []:
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Using configuration from
<install_dir>/bin/<apache_ManagedObject_name>/openssl.cnf
```

4 作成するキーファイルは，SSLCertificateKeyFile で指定した場所に移動してください。

5 認証ファイルは，SSLCertificateFile で指定した場所に移動してください。

メモ　テストであれば，ここで生成されるファイルで十分です。ただし，実用段階では，認知された第 2 章「セキュリティの概要」から承認された証明書を取得してください。

mod_ssl の有効性の確認

mod_ssl が動作していることを確認するには，https を使って Web サーバーにアクセスします。具体的な方法は，SSLEngine 指示文（72 ページの「キーファイルと証明書ファイルの作成」を参照）の配置場所によって異なります。

- SSLEngine がサーバーレベルで定義されている場合，すべてのサーバーアクセスが mod_ssl を経由します。
- SSLEngine がホストレベルで定義されている場合，特定のホストまでのアクセスは，正しいポート番号か IP アドレスで指定します。

  Web ブラウザからセキュリティで保護された接続を行うときは，URL で「http」ではなく「https」を使用します。次に例を示します。

  ```
  https://host.domain.com:443/index.html
  ```

各自の Web サーバーの情報については，mod_info を設定して設定情報を入手してください。

1 mod_info を有効にするには，次の場所にある httpd.conf ファイルを変更する必要があります。

  ```
  <install_dir>/var/domains/<domain_name>/configurations/<configuration_name>/
  mos/<apache_ManagedObject_name>/conf
  ```

  次の行のコメントを解除します。

  ```
  LoadModule info_module <install_dir>/lib/<apache_ManagedObject_name>/mod_info.so
  ```

2 さらに，次のコードセクションのコメントを解除します。

  ```
  <Location /server-info>
     SetHandler server-info
     Order deny, allow
  #   Deny from all
     Allow from .your_domain.com
  </Location>
  ```

重要　Deny from all 指示文のコメントは解除しないでください。

3 各自のドメインに合わせて Allow from .your_domain.com 指示文を変更します。

  Apache Web サーバーは，次のクエリーに応答してサーバーの設定情報を表示します。

  ```
  https://<server_name>/server-info
  ```

メモ　このクエリーは，セキュリティで保護された「https」接続を必要としません。mod_info の設定は，Apache Web サーバーが使用するプロトコルから独立しています。

Borland Web コンテナへの証明書パススルーの有効化

Apache Web サーバーの 71 ページの「mod_ssl に応じた Apache 設定ファイルの変更」ことによって，「https（SSL）」型の接続を処理できるように Web サーバーを設定できます。これにより，Apache Web サーバーはあらゆる SSL 認証情報を使用できるようになります。Borland Web コンテナが SSL 認証を管理するために，Apache Web サーバーと IIOP コネクタは SSL 認証情報を Borland Web コンテナを通して渡す必要があります。

VisiBroker の「証明書パススルー」機能を実装すると，Borland Web コンテナは，ブラウザと Web コンテナの間に Apache Web サーバーが存在していないかのように，ブラウザが提供するすべての SSL 情報にアクセスできます。さらに，Borland Web コンテナは SSL ベースの承認も制御できます。この機能によって，ブラウザと Borland Web コンテナの間に Apache Web サーバーが存在する場合も，Web アプリケーションは

CLIENT_CERT_
AUTH 認証メソッドを使用できます。

SSL 証明書およびその関連情報を「パススルー」するための Apache の設定

証明書パススルーを有効にするには，次の 2 つの手順を実行します。

1 ブラウザから渡された SSL 認証情報を共有環境にエキスポートするために，httpd.conf ファイルの 証明書パススルーのための httpd.conf ファイルの mod_ssl モジュールの設定します。

この共有環境によって，mod_iiop IIOP コネクタはその後に Borland Web コンテナに転送するためのデータを取得できます。

2 SSL 認証を転送するための httpd.conf ファイルの mod_iiop IIOP コネクタの設定して SSL 認証情報を Borland Web コンテナに転送します。

証明書パススルーのための httpd.conf ファイルの mod_ssl モジュールの設定

<install_dir>/var/domains/<domain_name>/configurations/<configuration_name>/mos/<apache_ManagedObject_name>/conf にある httpd.conf ファイルを変更して mod_ssl セクションの sslOptions +StdEnvVars +ExportCertData 指示文のコメントを解除します。次に例を示します。

```
<IfModule mod_ssl.c>
BrowserMatch ^Mozilla/[2345] nokeepalive
SSLEngine on
SSLRandomSeed startup builtin
SSLRandomSeed connect builtin
SSLCertificateFile C:¥BDP(b414)¥var¥domains¥base¥configurations¥j2eeSample/mos/
ApacheWebServer/conf/ssl.crt/server.crt
SSLCertificateKeyFile C:¥BDP(b414)¥var¥domains¥base¥configurations¥j2eeSample/
mos/ApacheWebServer/conf/ssl.key/server.key

# 次のコメントをはずすと，SSL 証明書と関連情報が有効になり，Apache 環境にエキ
スポートされる

#SSLOptions +StdEnvVars +ExportCertData

</IfModule>
```

SSL 認証を転送するための httpd.conf ファイルの mod_iiop IIOP コネクタの設定

証明書パススルー機能には，新しい IIOP 設定指示文が追加されました。IIopEnableSSLExport 指示文は，IIOP コネクタに SSL 要求を転送するように指示します。

IIopEnableSSLExport 指示文が有効で，クライアントの要求がセキュリティ保護に設定されている場合（https 型），IIOP コネクタは一連の環境変数の場所を探します。一連の環境変数が存在すると，IIOP コネクタは Borland Web コンテナにデータを要求の一部として転送します。

<install_dir>/var/domains/<domain_name>/configurations/<configuration_name>/mos/<apache_ManagedObject_name>/conf にある httpd.conf ファイルを変更してここに記載されている適切な指示文のコメントを解除します。

```
<IfDefine !ModIIOPNoAutoLoad>
LoadModule iiop2_module C:/BDP(b414)/lib/<apache_ManagedObject_name>/
mod_iiop2.dll
IIopLogFile C:¥BDP(b414)¥var¥domains¥base¥configurations¥j2eeSample/mos/
ApacheWebServer/logs/mod_iiop.log
IIopLogLevel error
IIopClusterConfig C:¥BDP(b414)¥var¥domains¥base¥configurations¥j2eeSample/mos/
ApacheWebServer/conf/WebClusters.properties
IIopMapFile C:¥BDP(b414)¥var¥domains¥base¥configurations¥j2eeSample/mos/
ApacheWebServer/conf/UriMapFile.properties

# 指示文：IIopLookupLocalRefs true | false
# 目的： ローカルの doc ツリーにリファレンスが見つかった場合，Apache が
UriMapFile マッピングを上書きできるようにします。
# デフォルト値：false - デフォルトでは，mod_iiop2 は一致参照を tomcat にディス
パッチする前にローカルリファレンスを検索しません。
#
#
#IIopLookupLocalRefs true
#
#
# 指示文：IIopChunkedUploading  true | false
# 目的： mod_iiop2 が POSTed データ（アップロード）を分割するかどうかを決定し
ます。Borland WebContainer の IIOP コネクタの enabledChunking 属性に対応しま
す。
# デフォルト値：false - Mod_iiop2 は，アップロードされたデータを分割しません。
#
#IIopChunkedUploading true
#
# 指示文：IIopUploadBufferSize n

# 目的：分割されたアップロードの一部として Borland Web Container に送信される
データを分割するサイズを mod_iiop が制御できるようにします。
IIopNoChunkedUploading が true に設定されている場合，この指示文は無視されます。
# デフォルト値：4096
#
#IIopUploadBufferSize 4096
#
# 指示文：IIopReapIdleConnections n

# 目的： すべての HTTP 要求の後に orb 接続プールからアイドル状態の接続を「n」
個取得するように VisiBroker orb に指示します。これにより，ネットワークリソース
が限られているシステムのパフォーマンスが向上します。
# デフォルト値：none - デフォルトでは，mod_iiop2 は orb にアイドル状態の接続を
取得するように要求しません。
#
#IIopReapIdleConnections 50

# 次の部分のコメントを解除して IIOP コネクタの証明書パススルー機能を有効にし
ます。
# メモ：mod_ssl モジュールの SSLOptions 指示文のコメントは，以前に解除してい
はずです。

#IIopEnableSSLExport true

</IfDefine>
```

Borland Web コンテナのセキュリティ

Web リソースのアクセスを保護するには，それらのリソースのセキュリティを確保する必要があります。VisiBroker を使用する Web リソースのセキュリティを確保するには，以下の手順にしたがって操作します。

1 Borland Web コンテナのセキュリティ保護

2 Web アプリケーションのセキュリティ確保

Borland Web コンテナのセキュリティ保護

デフォルトで，Borland Web コンテナは Borland セキュリティサービス領域（BSSRealm）を使用するように設定されています。Borland Web コンテナをセキュリティで保護するために，次の内容を実行する必要があります。

1 セキュリティをオン

2 認証

3 承認

Web アプリケーションのセキュリティ確保

VisiBroker Server では，アプリケーションのリソースを関連付ける URL を保護することで，Web アプリケーションごとにセキュリティを設定できます。Web アプリケーションのセキュリティを確保するには，まず保護する URL を選択します。URL を保護すると，有効なユーザー名とパスワードを入力しないユーザーはその URL をアクセスできなくなります。

Web リソースコレクション（サーブレット，JSP，HTML，Gif など）と関連付けられた URL について，保護対象が決まったら，以下の手順にしたがって操作します。

1 新しいセキュリティロールの定義：セキュリティロールにユーザーを割り当てます。これで，Web リソースをアクセスできるユーザーと，Web ブラウザで Web リソースを使用するときに可能な操作を設定します。

2 特定の Web リソースファイルのセキュリティ制限を定義：一定のサーブレットと JSP にマッピングする URL パターンを保護します。

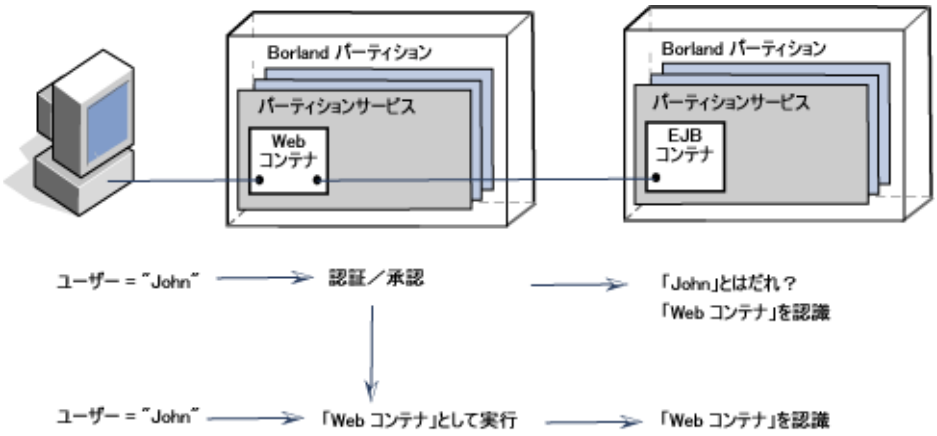
3 ログインの設定：URL パターンで，サーブレットと JSP のアクセスを制御するログインオプションを設定します。

たとえば，URL パターン jsp/security/servlet/*.jsp を収めた example.war ファイルの Web リソースをアクセスできる「開発者」セキュリティロールを設定するといった具合です。

3 階層の承認方式

クライアントブラウザから Borland Web コンテナにセキュリティを設定する方法のほか，3 階層承認方式を設定すれば，より複雑なクライアント／サーバー構造にも対応できます。3 階層の承認方式では，クライアントブラウザ，Web コンテナ，EJB コンテナを利用できます。

図 8.1　3 階層の承認方式



サーバー側には，2 つの異なるコンテナコンポーネントがあり，それぞれにセキュリティメカニズムがあります。したがって，ユーザー（John）がクライアント要求を送信すると，ユーザーのログイン ID に対して Borland Web コンテナレベルで認証と承認が行われます。

EJB コンテナ内の Bean にアクセスするために，Borland Web コンテナで稼動しているサーブレットがクライアント要求に必要だとします。ただし，EJB コンテナではユーザー「John」を認識できません。この EJB コンテナのセキュリティ機能を拡張するには，2 つの方法があります。

- 1 つは，EJB コンテナにすべてのユーザーの情報を与える方法です。
- 2 つ目の方法は，実行時のユーザー（run-as）で Web コンテナを稼動させる方法です。Web コンテナから EJB 呼び出しが行われると，Web コンテナは，EJB コンテナが認識するユーザー「として実行」（run-as）されます。Web アプリケーションは，実行時のユーザーが第 3 層のコンポーネントにアクセスするように設定できます。EJB を呼び出すサーブレットを持つ Web アプリケーションは，実行時のユーザーの「Web コンテナ」で設定します。この場合，実際のユーザーは「John」ですが，EJB コンテナはユーザーが「Web コンテナ」として認識します。

run-as ロールの設定

Borland Web コンテナは，Web アプリケーションのために実行時の設定をサポートします。Web アプリケーションは，ユーザーにマッピングされた run-as ロールで設定できます。

実行時のユーザーを設定するには，次の手順にしたがいます。

1. DDEditor で `.war` ファイルを開きます。
2. ナビゲーションペインで，`.war` ファイルを展開します。
3. サーブレットノードを選択します。
4. ［Properties］ペインの［General］タブにアクセスします。
5. ［Security Identity］フィールドで，ドロップダウンメニューから［Run as］を選択します。
6. 表示される［Run］セクションの［Descriptions］フィールドに［Run as］の簡単な説明を入力します。
7. ［Role］フィールドで，プルダウンメニューのリストから適切なロールを選択します。

メモ　run-as ロールを設定するほか，run-as のロールにマッピングされるプリンシパルも設定してください。詳細については，第 4 章「承認」を参照してください。

第 9 章

セキュリティプロパティ（Java）

プロパティ	説明	デフォルト値
vbroker.security.logLevel	このプロパティを使ってログ出力のレベルを制御します。0 はログ出力なし，7 は最大限のログ出力（デバッグメッセージ）を意味します。	0
vbroker.security.secureTransport	このプロパティは，転送の接続を暗号化するかどうかを制御します。true に設定すると，転送メッセージは暗号化されます。false に設定すると，暗号化されることなく送信されます。	true
vbroker.security.alwaysSecure	このプロパティは secureTransport プロパティとともにクライアント側のデフォルトの QoP を制御します。両方を true に設定すると，転送 QoP は SECURE_ONLY に設定され，クライアントはセキュアトランスポートだけを受け入れます。いずれかを false に設定すると，クライアントはトランスポート層にセキュリティを強制しません。	false
vbroker.security.server.transport	このプロパティは，サーバー側でサーバー転送 QoP を定義するために使用されます。指定できる値は，CLEAR_ONLY，SECURE_ONLY，または ALL です。これにより，CLEAR_ONLY または SECURE_ONLY を必要とするクライアントがサーバーに接続できるようになります。このプロパティは，secureTransport プロパティが true の場合にだけ有効です。	SECURE_ONLY
vbroker.security.disable	true に設定すると，すべてのセキュリティサービスが無効になります。	true
vbroker.security.transport.protocol	このプロパティは，セキュリティ転送プロトコルを選択するために使用します。指定できる値は，SSL，SSLv2，SSLv3，TLS，および TLSv1 です。このプロトコルについては，Sun Microsystem のマニュアル（http://java.sun.com/products/jsse/doc/guide/API_users_guide.html#SSC）を参照してください。	TLSv1
vbroker.security.requireAuthentication	サーバー側のみのプロパティは，クライアントが認証を受ける必要があるかどうかを指定するために使用します。	false
vbroker.security.enableAuthentication	サーバー側のみのプロパティ。この下位互換のプロパティは，PasswordBackEnd 形式の認証をサポートするために使用します。true に設定すると，プログラムは認証のために指定された PasswordBackEnd を構築します。	false

プロパティ	説明	デフォルト値
vbroker.security.authentication.callbackHandler	ユーザーと認証情報について対話するためのログインモジュールで使用するコールバックハンドラを指定します。次のいずれかを指定するか，または独自のカスタムコールバックハンドラを指定します。 com.borland.security.provider.authn.CmdLineCallbackHandler com.borland.security.provider.authn.HostCallbackHandler CmdLineCallbackHandler はパスワードのエコーオンで，HostCallbackHandler はパスワードのエコーオフです。	該当なし
vbroker.security.authentication.config	認証に使用する設定ファイルのパスを指定します。	null
vbroker.security.authentication.retryCount	リモート認証に失敗した場合の再試行回数です。	3
vbroker.security.authentication.clearCredentialsOnFailure	デフォルトでは，最大再試行回数の経過後に，認証領域で認証符号が不正であることが判明した場合，ORB が認証符号を保持します。最大再試行回数の経過後に，ORB に認証符号（認証情報）を消去させる場合は，このプロパティを true に設定します。	false
vbroker.security.login	このプロパティを true に設定すると，初期化時に vbroker.security.login.realms プロパティに記載されているすべての領域にログインを試みます。	false
vbroker.security.login.realms	ログインする領域のカンマで区切られたリストを提供します。このリストは，ログイン時に vbroker.security.login プロパティ（true に設定）または API ログイン（login() を使用）を通して使用されます。	該当なし
vbroker.security.vault	このプロパティは，ボールトファイルのパスを指定するために使用します。このプロパティは，vbroker.security.login が true または false のどちらに設定されている場合も有効です。	該当なし
vbroker.security.identity.reauthenticateOnFailure	true に設定すると，セキュリティサービスは CallbackHandler を使って認証情報の再取得を試みます。このプロパティには，適切なプロパティを使用するか，または実行時に適切なメソッドを呼び出してコールバックハンドラを設定する必要があります。	false
vbroker.security.identity.enableReactiveLogin	true に設定すると，セキュリティサービスは次のように動作します。サーバーがサポートするターゲットで，通信を試みているターゲットの ID が見つからない場合，セキュリティサービスはターゲットオブジェクトの IOR のいずれかのターゲットの認証情報を取得しようとします。このターゲットに対応する認証領域が使用できる（ユーザーが認証情報を提供する）場合は，ローカルにも認証を試みます。 リアクティブログインには，適切なプロパティを使用するか，または実行時に適切なメソッドを呼び出してコールバックハンドラを設定する必要があります。	true
vbroker.security.authDomains	使用できる承認ドメインのカンマで区切られたリストを指定します。次に例を示します。 vbroker.security.authDomains=<dom1>,<doma2>…	null
vbroker.security.domain.<domain_name>.rolemap_path	承認に使用すロールを記述する RoleDB ファイルの場所を指定します。スコープは，vbroker.security.authDomains に指定される <domain_name> の範囲になります。	該当なし
vbroker.security.domain.<domain_name>.rolemap_enableRefresh	true に設定すると，vbroker.security.domain.<domain_name>.rolemap_path プロパティに指定された RoleDB ファイルが動的にロードされます。動的ロードの間隔は，vbroker.security.domain.<domain_name>.rolemap_refreshTimeInSeconds プロパティによって指定します。	false
vbroker.security.domain.<domain_name>.rolemap_refreshTimeInSeconds	ロールマップのリフレッシュ時間を秒で指定します。	300

プロパティ	説明	デフォルト値
vbroker.security.domain.<domain name>.runas.<run_as_role_name>	run-as ロールの名前を指定します。 この値は，呼び出し元のプリンシパルを run-as ロールに含めるために use-caller-identity にするか，または run-as ロール名として run-as プリンシパルのエリアスを指定します。	該当なし
vbroker.security.peerAuthenticationMode	ピア認証モードを設定します。指定できる値は次のとおりです。 REQUIRE REQUIRE_AND_TRUST REQUEST REQUEST_AND_TRUST NONE JSSE の制限によって，REQUEST および REQUEST_AND_TRUST モードではピア証明書チェインを受け取ることができません。	NONE
vbroker.security.trustpointsRepository	信頼された証明書と CRL が入っているディレクトリのパスを指定するか，信頼された Keystore のパスを指定します。これらの値は TrustedCertificateEntry のインプリメンテーションになります。 デフォルト値は Directory:<path_to_certs> の形式のディレクトリ，または Keystore:<path_to_keystore> の形式の Keystore です。	該当なし
vbroker.security.defaultJSSETrust	true に設定すると，cacerts，jssecacerts などの JSSE デフォルト信頼ファイルが JRE にある場合は，それを使って信頼された証明書をロードします。	false
vbroker.security.assertions.trust.<n>	このプロパティは，信頼されたロールのリストを <role>@<authorization_domain> の形式で指定するために使用します。<n> は，信頼アサーションルールを連番として一意に識別するために使用します。 たとえば，vbroker.security.assertions.trust.1=ServerAdmin@default に設定すると，このプロセスは default 承認ドメインの ServerAdmin ロールによって生成されたすべてのアサーションを信頼します。	該当なし
vbroker.security.assertions.trust.all	true に設定すると，ピアが生成するすべてのアサーションを信頼します。	false
vbroker.security.server.requireUPIdentity	クライアントの認証（証明書ベースの認証かどうかに関係なく）のためにユーザー名／パスワードの組み合わせを送信することをサーバーが必要とする場合は，true に設定します。これはサーバー側のプロパティです。	該当なし
vbroker.security.cipherList	起動時にデフォルトで有効にする場合は，カンマで区切られた暗号のリストに設定します。設定しなければ，暗号スイートのデフォルトリストが有効になります。これらは，有効な SSL 暗号でなければなりません。	該当なし
vbroker.security.controlAdminAccess	セキュリティで保護されたサーバーでサーバーマネージャの操作を有効にする場合は true に設定します。	false
vbroker.security.serverManager.authDomain	vbroker.security.authDomains に記載されたセキュリティドメインをポイントします。指定されたドメインは，サーバーマネージャのロールベースのアクセスコントロールチェックに使用されます。ドメインのロールマップを指定する必要があります。	該当なし
vbroker.security.serverManager.role.all	サーバーマネージャのすべての操作にアクセスするために必要なロール名を指定します。	該当なし
vbroker.security.serverManager.role.<method_name>	サーバーマネージャの指定されたメソッドにアクセスするために必要なロール名を指定します。	該当なし
vbroker.security.support.gatekeeper.replyForSAS	このプロパティは，GateKeeper とともにセキュリティを有効にして使用します。true に設定すると，ユーザー名とパスワードは認証のためにバックエンドサーバーにデリゲートされません。	false

プロパティ	説明	デフォルト値
vbroker.security.domain.<domain_name>.defaultAccessRule	指定されたドメインのセキュリティロールがない場合に，デフォルトでドメインへのアクセスを許可するかどうかを設定します（grant または deny）。受け入れ可能な値は，grant または deny です。	grant
vbroker.se.iiop_tp.scm.ssl.listener.trustInClient	サーバー側のプロパティです。サーバーがクライアントに証明書を要求する場合は，true に設定します。また，サーバーは，サーバー側で信頼のプロパティを適切に設定することで，これらの証明書を信頼する必要があります。詳細については，vbroker.security.trustpointsRepository プロパティと vbroker.security.defaultJSSETrust プロパティを参照してください。	false
vbroker.security.wallet.type	ウォレットは，暗号化された秘密キーと各 ID の証明書チェインが入っているディレクトリの集合です。このプロパティによって，Directory:<path_to_identities> の形式を使用してすべての ID のディレクトリが入っているディレクトリをポイントできます。	該当なし
vbroker.security.wallet.identity	vbroker.security.wallet.type に定義されているパスに含まれ，特定の ID のキーと証明書情報が入っているディレクトリをポイントするために使用します。**メモ：**このプロパティの値には小文字しか使用できません。	該当なし
vbroker.security.wallet.password	秘密キーの暗号を解除するために使用するパスワードまたはログインに関連付けられたパスワードを指定します。	該当なし

第 10 章

セキュリティプロパティ（C++）

プロパティ	説明	デフォルト値
vbroker.security.logLevel	このプロパティを使ってログ出力のレベルを制御します。次の値を指定できます。LEVEL_WARN，LEVEL_NOTICE，LEVEL_INFO，および LEVEL_DEBUG 文字列。	LEVEL_WARN
vbroker.security.logFile	このプロパティを使ってログ出力をファイルにリダイレクトします。デフォルトのログ出力先は std::cerr です。	null
vbroker.security.secureTransport	このプロパティには，セキュアトランスポートをサポートするかどうかを設定します。false に設定すると，トランスポートは CLEAR_ONLY を使用します。	true
vbroker.security.alwaysSecure	これは，クライアント側のみのプロパティです。このプロパティは,セキュアトランスポートだけを使用するかどうかを決定します。 **メモ：**セキュアトランスポートだけを使用する場合は，secureTransport プロパティも true に設定する必要があります。	true
vbroker.security.server.transport	これは，サーバー側のみのプロパティです。このプロパティは，サーバーの転送を CLEAR_ONLY，SECURE_ONLY，または ALL のいずれかに定義します。このプロパティは，secureTransport プロパティが false に設定されている場合は無効です。	SECURE_ONLY
vbroker.security.disable	true に設定すると，すべてのセキュリティサービスが無効になります。	false
vbroker.security.requireAuthentication	サーバー側のみのプロパティです。クライアントを認証する必要があるかどうかを指定します。	true
vbroker.security.authentication.callbackHandler	ユーザーと認証情報について対話するためのログインモジュールのコールバックハンドラを指定します。次のいずれかを指定するか，または独自のカスタムコールバックハンドラを指定します。詳細については，第 11 章「VisiSecure for C++ API」を参照してください。 com.borland.security.provider.authn.CmdLineCallbackHandler com.borland.security.provider.authn.HostCallbackHandler CmdLineCallbackHandler はパスワードのエコーオンで，HostCallbackHandler はパスワードのエコーオフです。	HostCallbackHandler
vbroker.security.authentication.config	認証に使用する設定ファイルのパスを指定します。	null

プロパティ	説明	デフォルト値
vbroker.security.authentication.retryCount	リモート認証に失敗した場合の再試行回数です。	3
vbroker.security.login	このプロパティを true に設定すると，初期化時に vbroker.security.login.realms プロパティに記載されているすべての領域にログインを試みます。	true
vbroker.security.login.realms	ログインする領域のカンマで区切られたリストを提供します。このリストは，ログイン時に vbroker.security.login プロパティ(true に設定)または API ログインを通して使用されます。	該当なし
vbroker.security.vault	このプロパティは,ボールトファイルのパスを指定するために使用します。このプロパティは，vbroker.security.login が true または false のどちらに設定されている場合も有効です。	該当なし
vbroker.security.identity.reactiveLogin	true に設定すると，セキュリティサービスは次のように動作します。サーバーがサポートするターゲットで，通信を試みているターゲットの ID が見つからない場合,セキュリティサービスはターゲットオブジェクトの IOR のいずれかのターゲットの認証情報を取得しようとします。このターゲットに対応する認証領域が使用できる（ユーザーが認証情報を提供する）場合は，ローカルにも認証を試みます。 リアクティブログインには,適切なプロパティを使用するか,または実行時に適切なメソッドを呼び出してコールバックハンドラを設定する必要があります。デフォルトハンドラは HostCallbackHandler です。	true
vbroker.security.authDomains	使用できる承認ドメインのカンマで区切られたリストを指定します。次に例を示します。 vbroker.security.authDomains=domain1,domain2	該当なし
vbroker.security.domain.<domain-name>.rolemap_path	承認に使用すロールを記述する RoleDB ファイルの場所を指定します。スコープは，vbroker.security.authDomains に指定される <domain_name> の範囲になります。	該当なし
vbroker.security.domain.<domain_name>.rolemap_enableRefresh	true に設定すると，vbroker.security.domain.<domain_name>.rolemap_path プロパティに指定された RoleDB ファイルが動的にロードされます。動的ロードの間隔は，vbroker.security.domain.<domain_name>.rolemap_refreshTimeInSeconds プロパティによって指定します。	false
vbroker.security.domain.<domain_name>.rolemap_refreshTimeInSeconds	ロールマップのリフレッシュ時間を秒で指定します。	300

プロパティ	説明	デフォルト値
vbroker.security.peerAuthenticationMode	ピア認証モードを設定します。指定できる値は次のとおりです。 REQUIRE - ピア証明書は接続を確立する必要があります。ピアが証明書を提示しない場合は，接続が拒絶されます。ピア証明書は認証を受け，有効でない場合は，接続が拒絶されます。必要な場合は，この証明書を使ってトランスポート ID を確立できます。このモードでは，ピア証明書が信頼される必要はありません。 REQUIRE_AND_TRUST - ピア証明書が信頼される必要があり，そうでない場合は接続が拒絶されることを除き，REQUIRE モードと同じです。 REQUEST - ピア証明書が要求されます。ピアに証明書は必要ありません。ピアに証明書がない場合，トランスポート ID は確立されません。ただし，ピアが証明書を提示した場合は証明書が認証を受け，有効でない場合は接続が拒絶されます。必要な場合は，この証明書を使ってトランスポート ID を確立できます。このモードでは，ピア証明書が信頼される必要はありません。 REQUEST_AND_TRUST - ピア証明書が信頼される必要があり，そうでない場合は接続が拒絶されることを除き，REQUEST モードと同じです。 NONE - 認証は不要です。ハンドシェーク時に，証明書要求はピアに送信されません。接続は，ピアに証明書があるかどうかに関係なく受け入れられます。ピアにトランスポート ID はありません。	REQUIRE_AND_TRUST
vbroker.security.trustpointsRepository	信頼された証明書が入っているディレクトリのパスを指定します。Directory:<certs_dir> の形式で指定します。次に例を示します。 vbroker.security.trustpointsRepository=Directory:c:¥data¥identities¥Delta	該当なし
vbroker.security.assertions.trust.<n>	信頼されたロールのリストを <role>@<authorization_domain> の形式で指定するために使用します。<n> は，信頼アサーションルールを連番として一意に識別するために使用します。 たとえば，vbroker.security.assertions.trust.1=ServerAdmin@default に設定すると，このプロセスは default 承認ドメインの ServerAdmin ロールによって生成されたすべてのアサーションを信頼します。	該当なし
vbroker.security.assertions.trust.all	true に設定すると，ピアが生成するすべてのアサーションを信頼します。	false
vbroker.security.server.requireUPIdentity	サーバー側のみのプロパティです。クライアントの認証（証明書ベースの認証かどうかに関係なく）のためにユーザー名／パスワードを送信することをサーバーが必要とする場合は，true に設定します。vbroker.security.login.realms を設定すると，このプロパティは自動的に true になります。ただし，プロパティファイルで明示的に設定すると上書きできます。	該当なし
vbroker.security.cipherList	起動時にデフォルトで有効にする場合は，カンマで区切られた暗号のリストに設定します。設定しなければ，暗号スイートのデフォルトリストが有効になります。これらは，有効な SSL 暗号でなければなりません。	該当なし
vbroker.security.wallet.type	ウォレットは，暗号化された秘密キーと各 ID の証明書チェインが入っているディレクトリの集合です。このプロパティによって，Directory:<path_to_identities> の形式を使用してすべての ID のディレクトリが入っているディレクトリをポイントできます。	該当なし

プロパティ	説明	デフォルト値
vbroker.security.wallet.identity	vbroker.security.wallet.type に定義されているパスに含まれ，特定の ID のキーと証明書情報が入っているディレクトリをポイントします。	該当なし
vbroker.security.wallet.password	秘密キーの暗号を解除するために使用するパスワードまたはログインに関連付けられたパスワードを指定します。	該当なし
vbroker.security.CRLRepository	証明機関（CA）によって発行される失効した証明書（CRL：証明書失効リスト）のシリアル番号のリストを置くディレクトリを指定するために使用します。そのディレクトリのすべてのファイルがロードされ，（有効ではない）CRL として解釈されます。CRL ファイルは DER 形式でなければなりません。 CRL がロードされると，VisiSecure はハンドシェーク時にピアが送信するすべての証明書を調べます。ピアの証明書が CRL にある場合は，例外が生成されて接続が拒絶されます。詳細については，「セキュリティの概要」の第 2 章「セキュリティの概要」を参照してください。	該当なし

第 11 章

VisiSecure for C++ API

ここでは，VisiSecure for C++ バージョンで定義されている API について説明します。次のサブセクションがあります。

- 一般 API
- SSL と証明書 API
- QoP API
- 承認 API

すべてのクラスは，別の指定がない限り vbsec 名前空間にあります。

一般 API

一般 VisiSecure API では，Current と Context API について説明します。ここでは，プリンシパル，認証情報，およびサブジェクトに関する API 情報を提供します。さらに，Wallet API についても説明します。

class vbsec::Current

Current は，スレッド固有のセキュリティコンテキストのビューを表します。一部の呼び出しは，要求実行のコンテキストだけに適用されます。このオブジェクトは，次のコードを使って取得できます。

```
CORBA::Object_var obj = orb->resolve_initial_references("VBSecurityCurrent");
Current* c = dynamic_cast(obj.in());
```

インクルードファイル

このクラスを使用するには，vbsec.h ファイルをインクルードする必要があります。

メソッド

```
void asserting (const vbsec::Subject* caller)
```

呼び出し元の ID でサブジェクトをアサートします。

パラメータ	説明
caller	サブジェクトの呼び出し元です。

```
void clearAssertion ()
```

前の asserting の API 呼び出しによって作成されたアサーションをクリアします。アサーションが作成される前の呼び出し元が次の呼び出し元として復元されます。この API は，asserting とともに使用する必要があります。2 つのメソッドの呼び出しに不一致があると，予期しない呼び出し元 ID または予期しない例外が発生することがあります。

```
const vbsec::Subject* getPeerSubject ()
```

ピアサブジェクトにアクセスします。

戻り値 ピアを表す Subject オブジェクトへのポインタ。

```
const vbsec::Subject* getCallerSubject ()
```

呼び出し元サブジェクトにアクセスします。

戻り値 呼び出し元を表す Subject オブジェクトへのポインタ。

```
const vbsec::SSLSession* getPeerSession (CORBA::Object* peer)
```

SSLSession ピアを取得します。この呼び出しは，この要求のクライアントピアの SSLSession を返します。このメソッドを要求のコンテキストの外部で呼び出すことはできません。

パラメータ	説明
peer	バインドから取得した peer オブジェクトです。

戻り値 確立されている SSLSession へのポインタ。

例外 このメソッドが要求コンテキストの外部で呼び出されるか，またはクリア TCP 接続を通して要求を受け取る要求コンテキストで呼び出されると BAD_OPERATION が生成されます。

class vbsec::Context

Context は，クライアントが実行するセキュリティコンテキストを表します。このクラスは，次のコードを使って取得できます。

```
CORBA::Object_var obj = orb->resolve_initial_references("VBSecurityContext");
Context* c = dynamic_cast(obj.in());
```

インクルードファイル

このクラスを使用するには，vbsec.h ファイルをインクルードする必要があります。

メソッド

```
void login()
```

システムにログインします。vbroker.security.loginRealms プロパティに定義された領域にログインします。指定された領域のリストを参照して各領域の認証を受けます。

```
void login (vbsec::CallbackHandler& handler)
```

指定された CallbackHandler を使ってログイン情報を取得してシステムにログインします。

パラメータ	説明
handler	情報を取得するために使用するデフォルトのコールバックハンドラです。

```
void login (const std::string& realm)
```

特定の領域のシステムにログインします。

パラメータ	説明
realm	ログインする領域です。

```
void login (const std::string& realm, vbsec::CallbackHandler& handler)
```

指定されたコールバックハンドラを使って情報を取得し，指定された領域のシステムにログインします。

パラメータ	説明
realm	ログインする領域です。
handler	情報を取得するために使用するデフォルトのコールバックハンドラです。

```
void login (const vbsec::Wallet& wallet)
```

wallet を使ってシステムにログインします。ウォレットは，WalletFactory API を使って作成します。

パラメータ	説明
wallet	ログインに使用するウォレットです。

```
void login (const std::vector<const vbsec::Wallet*>& wallet)
```

ベクタとして指定された一連のウォレットを使ってシステムにログインします。

パラメータ	説明
wallet	ログインに使用するウォレットです。

```
const vbsec::Subject* getSubject (const std::string& realm)
```

指定された領域に対応する Subject を取得します。

パラメータ	説明
realm	プリンシパルの領域です。

戻り値 領域のサブジェクトを表す Subject オブジェクトへのポインタ。

```
void loadVault (std::istream& stream, const CSI::UTF8String& vaultPass)
```

指定さられたボールトをロードします。ボールト内の ID がシステムにロードされます。このメソッドを使用する場合，ログインは必要ありません。

パラメータ	説明
stream	ボールト情報をバイナリ形式で読み出すストリームです。
vaultPass	ボールト情報の暗号を解除するためのパスワードです。

```
void logout()
```

すべての領域からユーザーをログアウトします。

```
void logout (const std::string& realm)
```

指定された領域からユーザーをログアウトします。

パラメータ	説明
realm	ログアウトする領域です。

```
void setCallbackHandler (vbsec::CallbackHandler* handler)
```

デフォルトコールバックハンドラをプログラムによって設定します。これは，vbroker.security.authentication.callbackHandler プロパティを使用する場合とほとんど同じです。

パラメータ	説明
handler	設定する CallbackHandler です。

```
void generateVault( std::ostream& stream, const CSI::UTF8String& password)
```

ボールトを生成します。ボールトが渡されたストリームに書き出され，指定されたパスワードを使って暗号化されます。このパスワードは，ボールトの暗号を解除するためにも使用します。パスワードに null を指定することもできます。ボールトには，システムのすべての ID が入っています。

パラメータ	説明
stream	ボールト情報をバイナリ形式で書き込むストリームです。
password	ボールト情報を暗号化するために使用するパスワードです。

```
vbsec::Subject* authenticateUser (const vbsec::Wallet& wallet)
```

指定されたウォレットの認証情報を認証します。ログインはウォレットを使って実行されますが，システム ID の 1 つとして認証されたサブジェクトは使用されません。

パラメータ	説明
wallet	認証に使用するウォレットです。

```
vbsec::Subject* importIdentity (const vbsec::Wallet& wallet)
```

指定されたウォレットの認証情報を使ってサブジェクトをインポートします。このメソッドにログインは必要ありません。サブジェクトは，システム ID の 1 つとして使用されません。

パラメータ	説明
wallet	インポートする ID に対応するウォレットです。

```
void setPRNGSeed (const CORBA::OctetSequence& seed)
```

SSL 層で使用する擬似ランダムジェネレータのための seed を設定します。

パラメータ	説明
seed	PRNG のシードです。

```
ssl::CipherSuiteInfoList* listAvailableCipherSuites()
```

SSL 層で使用できる暗号化スイートのリストを取得します。これは，**利用可能な**すべての暗号化スイートが有効になっているわけではないという点で getEnabledCipherSuites 呼び出しとは異なります。

戻り値 利用可能であっても SSL 層での使用が有効になっていない暗号化スイートのリスト。

```
void enableCipherSuites (const ssl::CipherSuiteInfoList& suites)
```

すべての SSL セッションに対して有効にする必要がある暗号化スイートを設定します。

パラメータ	説明
suites	IDL が生成する CipherSuiteInfoList 型

```
ssl::CipherSuiteInfoList* getEnabledCipherSuites()
```

すべての SSL セッションに対して有効になっている暗号化スイートの集合を取得します。

戻り値 すべての SSL セッションに対して有効になっている暗号化スイート。

```
void setSSLContext (vbsec::VBSSLContext* ctx)
```

SSL コンテキストを設定します。これにより，VBSSLContext で定義されている情報を使って SSL セッションを確立できます。VBSSLContext は，SecureSocketProvider API を使って作成します。

パラメータ	説明
ctx	すべての SSL セッションの確立に使用される VBSSLContext です。

```
VBSSLContext& getSSLContext()
```

setSSLContext() を使って設定される VBSSLContext を取得するか，またはデフォルトの VBSSLContext オブジェクトを返します。

戻り値 すべての SSLSession の確立に使用される VBSSLContext です。

class vbsec::Principal

プリンシパルは，ユーザーの ID を表します。これは仮想クラスです。

インクルードファイル

このクラスを使用するには，vbsec.h ファイルをインクルードする必要があります。

メソッド

```
std::string getName() const
```

戻り値 プリンシパルの名前。

```
std::string toString() const
```

プリンシパルの文字列表現を取得します。

戻り値 プリンシパルの文字列表現です。

class vbsec::Credential

認証情報は，ユーザー名とパスワードなどの身元を認証するための情報を表します。これは仮想クラスです。

インクルードファイル

このクラスを使用するには，vbsec.h ファイルをインクルードする必要があります。

class vbsec::Subject

サブジェクトは，人物などの単一のエンティティに関する情報のグループ化を表します。この種の情報には，サブジェクトの身元，およびパスワードや暗号キーなどのセキュリティに関する属性があります。

インクルードファイル

このクラスを使用するには，vbsec.h ファイルをインクルードする必要があります。

メソッド

```
Principal::set& getPrincipals()
```

サブジェクトのプリンシパルを取得します。

戻り値 サブジェクトのプリンシパルの集合。この集合の内容を変更してもサブジェクトには影響しません。

```
void clearPrincipals()
```

サブジェクトからプリンシパルをクリアします。サブジェクトのすべてのプリンシパルが削除されます。

```
Credential::set& getPublicCredentials()
```

サブジェクトの公開認証情報を取得します（公開キーなど)。

戻り値 サブジェクトの公開認証情報の集合。この集合の内容を変更してもサブジェクトには影響しません。

```
void clearPublicCredentials()
```

サブジェクトの公開認証情報をクリアします。サブジェクトのすべての公開認証情報が破棄されて削除されます。

```
Credential::set& getPrivateCredentials()
```

サブジェクトの秘密認証情報を取得します（秘密キーなど)。

戻り値 サブジェクトの秘密認証情報の集合。この集合の内容を変更してもサブジェクトには影響しません。

```
void clearPrivateCredentials()
```

サブジェクトの秘密認証情報をクリアします。サブジェクトのすべての秘密認証情報が破棄されて削除されます。

```
Principal::set getPrincipals (const type_info& info) const
```

提供されている実行時の型と同じ型情報を持っているサブジェクトのプリンシパルの集合を取得します。

パラメータ	説明
info	返されたプリンシパルが持っている実行時の型情報です。

戻り値 指定された実行時の型と同じ型情報を持っているサブジェクトのプリンシパルの集合。この集合の内容を変更してもサブジェクトには影響しません。

```
Credential::set getPublicCredentials (const type_info& info) const
```

提供されている実行時の型と同じ型情報を持っているサブジェクトの公開認証情報の集合を取得します。

パラメータ	説明
info	返された公開認証情報が持っている実行時の型情報です。

戻り値 指定された実行時の型と同じ型情報を持っているサブジェクトの公開認証情報の集合。この集合の内容を変更してもサブジェクトには影響しません。

```
Credential::set getPrivateCredentials (const type_info& info) const
```

提供されている実行時の型と同じ型情報を持っているサブジェクトの秘密認証情報の集合を取得します。

パラメータ	説明
info	返された秘密認証情報が持っている実行時の型情報です。

戻り値 指定された実行時の型と同じ型情報を持っているサブジェクトの秘密認証情報の集合。この集合の内容を変更してもサブジェクトには影響しません。

class vbsec::Wallet

Wallet は，一般にログイン API 呼び出しで使用される認証情報の所有者です。Wallet は，WalletFactory API を使って作成され，複数の種類の認証情報が入っています。

インクルードファイル
このクラスを使用するには，vbsec.h ファイルをインクルードする必要があります。

メソッド

```
std::string getTarget () const
```

ウォレットが認証を受けるターゲットを取得します。

戻り値 ターゲット情報の文字列表現。

```
void populateSubject (Subject& subject)
```

認証のための認証情報またはその他の情報を指定されたサブジェクトに格納します。

パラメータ	説明
subject	ウォレットがデータを格納するサブジェクトです。

class vbsec::WalletFactory

WalletFactory は，複数の型のウォレットを作成するファクトリクラスです。

インクルードファイル
このクラスを使用するには，vbsec.h ファイルをインクルードする必要があります。

メソッド

```
Wallet* createCertificateWallet (const std::string& name,
                                 const std::string& password,
                                 const std::string& alias,
                                 const std::string& keypassword,
                                 short usage)
```

C++ キーストアを使って証明書ウォレットを作成します。C++ キーストアは Java キーストアとほとんど同じですが，ディレクトリ構造を使って実装されます。この API によって作成されたウォレットを使ってログインすると，SSL 層で証明書チェインが使用されます。

パラメータ	説明
name	キーストアのディレクトリです。
password	キーストアのパスワードです（**このリリースでは使用しません**）。
alias	キーストアで使用するエリアスです。
keypassword	指定されたエリアスの秘密キーのパスワードです。
short usage	証明書情報の使用状況です（CLIENT，SERVER，または ALL）。

戻り値 指定された情報が入っている証明書ウォレット。

```
Wallet* createCertificateWallet (const CORBAsec::X509CertList& chain,
                                 const CORBAsec::ASN1Object& privkey,
                                 const CSI::UTF8String& password)
```

証明書チェイン，秘密キー，およびパスワードを使って証明書ウォレットを作成します。

パラメータ	説明
chain	ウォレットを作成するための証明書チェインです。
privkey	証明書チェインの秘密キーです。
password	秘密キーのパスワードです。

戻り値 指定された情報が入っている証明書ウォレット。

```
Wallet* createIdentityWallet (const std::string& username,
                              const std::string& password,
                              const std::string& realm)
```

ユーザー名，パスワード，およびウォレットが認証を受ける領域を使って ID ウォレットを作成します。

パラメータ	説明
username	ID のユーザー名です。
password	ID のパスワード。
realm	ウォレットが認証を受ける領域です。

戻り値 指定された情報が入っている ID ウォレット。

```
Wallet* createIdentityWallet (const std::string& username,
                              const std::string& password,
                              const std::string& realm,
                              const std::vector<std::string>& groups)
```

ユーザー名，パスワード，ウォレットが認証を受ける領域，および一連のグループ属性を使って ID ウォレットを作成します。

パラメータ	説明
username	ID のユーザー名です。
password	ID のパスワード。
realm	ウォレットが認証を受ける領域です。
groups	ID が所属する一連のグループ属性です。

戻り値 指定された情報が入っている ID ウォレット。

SSL API

ここでは，VisiSecure の SSL インプリメンテーションと対話するさまざまな SSL API について説明します。

class vbsec::SSLSession

`SSLSession` は，現在の SSL 接続のセッションを表します。`SSLSession` は，`vbsec::Context using getPeerSession()` から取得できます。

インクルードファイル

このクラスを使用するには，`vbssp.h` ファイルをインクルードする必要があります。

メソッド

```
time_t getEstablishmentTime() const
```

SSL 接続が確立された時刻を取得します。

戻り値 SSL 接続が確立された時刻。

```
const ssl::CipherSuiteInfo& getNegotiatedCipher() const
```

指定された SSL 接続に対するピアからネゴシエートされた暗号を返します。

戻り値 指定された SSL 接続に対するピアからネゴシエートされた暗号。

```
const CORBAsec::X509CertList& getPeerCertificates() const
```

ピアの証明書チェインを取得します。

戻り値 ピア証明書チェイン。

```
const CORBAsec::X509Cert* getTrustpoint() const
```

ピアが信頼されている信頼ポイントを取得します。ピアに証明書がないか，または信頼されていない場合は null を返します。

戻り値 ピアが信頼されている信頼ポイントまたは null。

```
char* getPeerAddress() const
```

ピアの IP アドレスを取得します。

戻り値 xxx.xx.xx.xx 形式の文字列によるピア IP アドレス。

```
CORBA::UShort getPeerPort() const
```

この接続が使用するピアのポート番号を返します。

戻り値 接続におけるピアのポート番号。

```
void prettyPrint (std::ostream& os) const
```

指定された出力ストリームに SSLSession 情報を出力します。

パラメータ	説明
os	SSLSession 情報を出力する出力ストリームです。

class vbsec::VBSSLContext

VBSSLContext には，SSLSession を確立するために必要な情報が入っています。このオブジェクトは，SecureSocketProvider::createSSLContext() を使って作成します。

インクルードファイル

このクラスを使用するには，vbssp.h ファイルをインクルードする必要があります。

メソッド

```
const CORBAsec::X509CertList& getCertificates() const
```

SSL 層で使用する ID を表す証明書チェインを取得します。

戻り値 SSL 層で使用する ID を表す証明書チェイン。

```
void setCipherSuiteList (const ssl::CipherSuiteInfoList& list)
```

このメソッドは，SSL 接続で使用できる暗号を指定するために使用します。

パラメータ	説明
list	SSL 接続で使用できる暗号のリストです。

```
const ssl::CipherSuiteInfoList& getCipherSuiteList() const
```

現在 SSL 層が使用している暗号を返します。

戻り値 現在 SSL 層が使用している暗号。

```
void addTrustedCertificate (const CORBAsec::X509Cert_var& trusted)
```

プログラムによって信頼された証明書を SSL コンテキストに追加します。

パラメータ	説明
trusted	信頼される証明書です。

```
CORBAsec::X509CertList* getTrustedCertificates() const
```

信頼されている証明書のリストを取得します。

戻り値 信頼されている証明書のリスト。

class ssl::CipherSuiteInfo

CipherSuiteInfo は，2 つのフィールドを含む構造体です。

- CORBA::ULong SuiteID
- CORBA::String_var Name

この IDL 構造体には，SSL 仕様に基づく暗号化を記述した 2 つのフィールドがあります。SuiteID 値のリストとそれらの名前は，インクルードファイル ssl_c.h にあります。

インクルードファイル

このクラスを使用するには，ssl_c.hh ファイルをインクルードする必要があります。

class CipherSuiteName

このクラスは，セキュリティサービスが使用する暗号化に関する情報を提供します。

インクルードファイル

このクラスを使用するには，csstring.h ファイルをインクルードする必要があります。

メソッド

```
static const char* toString (int tag)
```

サポートされている SSL 暗号化の標準表現を返します。

パラメータ	説明
tag	暗号名に関連付けられたタグです。

戻り値 暗号化の文字列化表現を返します。

```
static const int fromString (char* description)
```

指定された暗号の説明に関連付けられたタグを返します。

パラメータ	説明
説明	暗号の文字列化表現です。

戻り値 引数として提供された暗号名に関連付けられたタグ。

class vbsec::SecureSocketProvider

SecureSocketProvider は，セキュアソケット接続のプロバイダです。SSL コンテキストの作成，SSL 証明書の処理，セキュアソケットに関連するその他の情報の管理を行うための機能を提供します。

インクルードファイル

このクラスを使用するには，vbssp.h ファイルをインクルードする必要があります。

メソッド

```
vbsec::VBSSLContext* createSSLContext (const CORBAsec::X509CertList& chain,
                                       const CORBAsec::ASN1Object& privkey,
                                       const CSI::UTF8String& password)
```

このメソッドは，指定された情報を使って SSL コンテキストを作成します。その後に SSL コンテキストは vbsec::Context に渡され，SSL 接続を確立するために使用されます。

パラメータ	説明
chain	証明書チェインです。
privkey	秘密キーのオブジェクトです。
password	秘密キーのパスワードです。

戻り値 指定された情報が入っている VBSSLcontext。

```
void setPRNGSeed (const ssl::Current::PRNGSeed& seed)
```

SSL 層で使用する擬似ランダム番号ジェネレータのための seed を設定します。

パラメータ	説明
seed	PRNG のシードです。

```
const ssl::CipherSuiteInfoList& listAvailableCipherSuites() const
```

SSL 層で使用できる暗号化スイートのリストを取得します。これは，利用可能なすべての暗号化スイートが有効になっているわけではないという点で getEnabledCipherSuites 呼び出しとは異なります。

戻り値 利用可能であっても SSL 層での使用が有効になっていない暗号化スイートのリスト。

```
const CertificateFactory& getCertificateFactory() const
```

証明書ファクトリを取得します。

戻り値 CertificateFactory オブジェクト。

class ssl::Current

ssl::Current を使用して，クライアントアプリケーションやサーバーオブジェクトを秘密キーに設定し，証明書情報をピアに提供します。さらに，このインターフェースを使って SSL 接続を設定し，証明書と秘密キーを SSL 接続で関連付けます。

秘密キーと証明書にはヘッダーとトレーラ行が含まれており，キーや証明書の開始部分と終了部分を示しています。このインターフェースが秘密キーと証明書チェインに対して提供するすべてのメソッドは，これらのヘッダーとトレーラ行が必要になります。これらの行に対する解析規則は次のとおりです。

- 証明書に対して認識されたヘッダーの行形式は次のとおりです。

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

- 秘密キーに対して認識されたヘッダーの行形式は次のとおりです。

  ```
  -----BEGIN ENCRYPTED PRIVATE KEY-----
  -----BEGIN RSA PRIVATE KEY-----
  ```

- すべてのヘッダー行は，新しい行を示す文字で終わる必要があります。
- すべてのトレーラ行は，行の始まりと終わりに新しい行を示す文字が必要です。PEM 形式の秘密キーには，Proc-Type と DEK-Info という 2 つの追加ヘッダー行があり，ほかの秘密キーにはありません。両方の行が存在し，行の終わりに新しい行を示す文字が必要です。

このオブジェクトは，次のコードを使って取得できます。

```
CORBA::Object_var obj = orb->resolve_initial_references("SSLCurrent");
ssl::Current_var current = ssl::Current::_narrow(obj);
```

インクルードファイル

このクラスを使用するには，ssl_c.hh ファイルをインクルードする必要があります。

メソッド

```
CORBA::ULong getNegotiatedCipher(CORBA::Object_ptr peer)
```

指定された SSL 接続に対するピアからネゴシエートされた暗号を返します。

パラメータ	説明
peer	ネゴシエートされた暗号から取得するピアです。

戻り値 使用された暗号を示す値（タグ）。（文字列表現を取得するには，CipherSuiteName::toString を使用します）。

例外 CORBA::BAD_OPERATION：オブジェクトが null の場合または接続が SSL を使用していない場合。

```
CORBAsec::X509CertList_ptr getPeerCertificateChain(CORBA::Object_ptr peer)
```

ピアの認証チェインを取得します。サーバーからの情報を取得するのに通常はクライアントアプリケーションによって起動されますが，サーバーはクライアントから情報を要求することもできます。

パラメータ	説明
peer	ネゴシエートされた暗号から取得するピアです。

戻り値 使用された暗号を示す値。（文字列表現を取得するには，CipherSuiteName::toString を使用します）。

例外 CORBA::BAD_OPERATION：オブジェクトが null の場合または接続が SSL を使用していない場合。

```
char* getPeerAddress(CORBA::Object_ptr peer)
```

この接続で使用しているソケットパラメータの記述を返します。

パラメータ	説明
peer	情報から取得するピアです。

戻り値 xxx.xx.xx.xx 形式の文字列によるピア IP アドレス。

例外 CORBA::BAD_OPERATION：オブジェクトが null の場合または接続が SSL を使用していない場合。

```
CORBA::Boolean isPeerTrusted(CORBA::Object_ptr peer)
```

ピアの証明書チェインが信頼できるかどうかをテストします。チェインの証明書のいずれかが trustpoint にある場合は信頼できます。

パラメータ	説明
peer	情報から取得するピアです。

戻り値 チェインが信頼されている場合は true，されていない場合は false。

例外 CORBA::BAD_OPERATION：オブジェクトが null の場合または接続が SSL を使用していない場合。

```
trust::Trustpoints_ptr Trustpoints getTrustpointsObject()
```

trustpoint リポジトリへのリファレンスを返します。この API を使って trustpoint オブジェクトへアクセスし，trustpoint を設定します。

戻り値 _var に割り当てる必要がある trustpoint リポジトリへのリファレンス。

```
void setPRNGSeed (const ssl::Current::PRNGseed& seed)
```

SSL 層で使用する擬似ランダム番号ジェネレータのための seed を設定します。

パラメータ	説明
seed	PRNG の OctetSequence シードです。

```
void setPKprincipal (const CORBAsec::ASN1ObjectList chain,&
                     const CORBAsec::ASN1Object& privkey,&
                     const char* password);
```

このメソッドを使用することで，クライアントやサーバーで SSL 接続で使用する必要がある認証チェインと秘密キーを設定できます。これは，サーバーでは必須，クライアントではオプションです。第 10 章「セキュリティプロパティ（C++）」の peerAuthenticationMode プロパティも参照してください。

パラメータ	説明
chain	認証チェイン。
privkey	SSL 接続で使用する秘密キー。
password	秘密キーのパスワードです。

例外 CORBA::BAD_PARAM（ユーザー名またはパスワードが null の場合）。

```
void setCipherSuiteList (const ssl::CipherSuiteInfoList& list)
```

このメソッドを使用することで，クライアントやサーバーで SSL 接続で利用可能な暗号を指定できます。

パラメータ	説明
list	カンマで区切られた暗号スイートのリストです。

```
ssl::CipherSuiteInfoList* listAvailableCipherSuites()
```

VisiSecure で使用できる暗号スイートのリストを返します。不要になったメモリは解放する必要があります。

戻り値 暗号スイートのリスト。

```
ssl::CipherSuiteInfoList* getCipherSuiteList()
```

現在 SSL 層が使用している暗号を返します。

戻り値 暗号スイートのリスト。

```
void setP12Identity (const CORBASEC::ASNIOBJECT& pks12cert, const char*
password)
```

パラメータ	説明
pks12cert	PKCS#12 形式のデータ。
password	秘密キーのパスワード。

証明書 API

この API には，証明書で使用するクラスとメソッドが入っています。

class vbsec::CertificateFactory

証明書とキーを処理するためのユーティリティクラスです。

インクルードファイル

このクラスを使用するには，vbssp.h ファイルをインクルードする必要があります。

メソッド

```
CORBAsec::X509CertList* importCertificateChain (const CORBAsec::ASN1ObjectList&
certs) const
```

CORBAsec::ASN1ObjectList 形式の証明チェインを CORBAsec::X509CertList にインポートします（VBSSLContext で使用します）。

パラメータ	説明
certs	証明書チェインの ASN1ObjectList 形式の表現です。

戻り値 CORBA 転送のための証明書チェインの CORBAsec::X509CertList 形式の表現。

```
CORBAsec::X509CertList* importCertificates (const CORBAsec::ASN1ObjectList&
certs) const
```

証明書リストを CORBAsec::ASN1ObjectList の形式で CORBAsec::X509CertList にインポートします。証明書は相互に関連付けられている必要はありません。インポート後にも元の順序は保持されます。

パラメータ	説明
certs	証明書リストの ASN1ObjectList 形式の表現です。

戻り値 証明書リストの CORBAsec::X509CertList 形式の表現です。

```
CORBAsec::ASN1Object* importPrivateKey (const CORBAsec::ASN1Object& key) const
```

秘密キーを BASE64 または PEM 形式から DER 形式に変換します。

パラメータ	説明
key	秘密キーオブジェクトの ASN1ObjectList 形式の表現です。

戻り値 秘密キーの DER 形式。

```
CORBAsec::X509CertList* importCertificateChain (const CORBAsec::ASN1Object&
pkcs12bytes,
                                                const CSI::UTF8String&
password) const
```

証明書チェインを pkcs12 バイナリからインポートします。

パラメータ	説明
pkcs12bytes	pkcs12 バイナリの ASN1ObjectList 形式の表現です。
password	pkcs12 バイナリのパスワードです。

戻り値 証明書チェインの CORBAsec::X509CertList 形式の表現です。

```
CORBAsec::ASN1Object* importPrivateKey (const CORBAsec::ASN1Object&
pkcs12bytes,
                                        const CSI::UTF8String& password) const
```

秘密キーを pkcs12 バイナリからインポートします。

パラメータ	説明
pkcs12bytes	pkcs12 バイナリの ASN1ObjectList 形式の表現です。
password	pkcs12 バイナリのパスワードです。

戻り値 秘密キーオブジェクトの CORBAsec::ASN1Object 形式の表現です。

```
const CertificateFactory& printCertificate (const CORBAsec::X509Cert&
certificate, std::ostream& stream) const
```

証明情報を出力ストリームに出力します。

パラメータ	説明
certificate	出力する証明書です。
stream	出力するストリームです。

戻り値 CertificateFactory を返します。

```
bool passwordForPrivatekey (const CSI::UTF8String& password,  const
CORBAsec::ASN1Object& privkey) const
```

指定されたパスワードが指定された秘密キーオブジェクトの暗号を解除できるかどうかをテストします。

パラメータ	説明
password	テストするパスワードです。
privkey	暗号を解除する秘密キーオブジェクトです。

戻り値 暗号の解除に成功すると true，失敗すると false。

class CORBAsec::X509Cert

このクラスは X509 証明書を表します。クライアントアプリケーションが CORBA オブジェクトにバインドされると，クライアントはこのインターフェースを使ってサーバーの証明書情報を取得します。クライアントに証明書がある場合，サーバーはこのインターフェースを使ってクライアントの証明書情報を取得することができます。

インクルードファイル

このクラスを使用するには，X509Cert_c.hh ファイルをインクルードする必要があります。

メソッド

```
char* getSubjectDN()
```

証明書に入っているサブジェクト DN を返します。

戻り値 サブジェクト名が次のような形式で返されます。

```
CN=<value>, OU=<value>, O=<value>, L=<value>, S=<value>, C=<value>
```

```
char* getIssuerDN()
```

証明書に入っている発行元 DN を返します。

戻り値 サブジェクト名が次のような形式で返されます。

```
CN=<value>, OU=<value>, O=<value>, L=<value>, S=<value>, C=<value>
```

```
CORBA::OctetSequence * getSignatureAlgorithm()
```

証明書に含まれている署名アルゴリズムを返します。

戻り値 証明書に含まれている署名アルゴリズム。

```
CORBA::OctetSequence * getHash(CORBASEC::HashAlgorithm algorithm)
```

証明書のハッシュを返します。

パラメータ	説明
algorithm	ハッシュアルゴリズムです。指定できる値は，CORBASec::MD5，CORBASec::MD2，および CORBASec::SHA1 です。

戻り値 指定されたアルゴリズムを使用した証明書のハッシュ。

```
CORBAsec::ASN1Object_ptr getDER()
```

この証明書の DER 暗号化形式を返します。

戻り値 この証明書の ASN.1 DER 暗号化形式（_var に割り当てられます）。

```
CORBAsec::SerialNumberValue_ptr getSerialNumber()
```

証明書のシリアル番号を返します。

戻り値 証明書のシリアル番号。

```
CORBAsec::X509CertExtensionList_ptr getExtensions()
```

X509CertExtension のリストとして，この証明書で利用可能なすべての拡張を返します。

戻り値 X509CertExtension のリストとして，この証明書で利用可能なすべての拡張を返します。または，この証明書に拡張がない場合は，Null 長の配列を返します。拡張は解析できません。

```
CORBA::Boolean isValid (CORBA::ULong_out date)
```

この証明書の日付が，有効な開始日と終了日の間かどうかをチェックします。

パラメータ	説明
date	UNIX の時刻形式を使って証明書の有効期限を設定する out 引数。

戻り値 証明書が有効な場合は true，無効な場合は false 。

```
CORBA::ULong startDate()
```

証明書の有効期限の開始日を取得します。

戻り値 1970 年 1 月 1 日 0 時からの秒数を表す int を返します。

```
CORBA::ULong endDate()
```

証明書の有効期限日を取得します。

戻り値 1970 年 1 月 1 日 0 時からの秒数を表す int を返します。

```
CORBA::Boolean equals (CORBAsec::X509Cert_ptr other)
```

2 つの CORBAsec::X509Cert 証明書を比較します。

パラメータ	説明
other	この証明書と比較する証明書です。

戻り値 2 つの証明書が同じ場合は true (1UL)，そうでなければ false (0UL) を返します。

```
CORBA::Boolean isTrustpoint()
```

この証明書が trustpoint かどうか，その場合はそれが信頼できる証明書かどうかを確認します。

戻り値 証明書が trustpoint の場合は，true を返します。

class CORBAsec::X509CertExtension

このクラスは，次のような X509 証明書拡張を示す IDL 構造体です。

```
struct X509CertExtension {
long seq;
sequence<long> oid;
boolean critical;
sequence<octet> value;
};
```

パラメータ	説明
seq	証明書拡張の一意の番号です。
oid	拡張の oid です。
value	oid に指定されている形式によってエンコードされた拡張の値です。

インクルードファイル

このクラスを使用する場合は，X509Cert_c.hh ファイルをインクルードする必要があります。

QoP API

ここでは，VisiSecure が提供する保護品質（QoP）API について説明します。

class vbsec::ServerConfigImpl

ServerConfigImpl は，csiv2::ServerQoPConfig のインプリメンテーションで，次のような IDL 構造体です。

```
ServerConfigImpl (
    CORBA::Boolean disable,
    CORBA::Short transport,
    CORBA::Boolean trustInClient,
    csiv2::AccessPolicyManager* access_manager,
    const CORBA::StringSequence& realms = _available,
    CORBA::Short requiredIdentityType = csiv2::ServerQoPConfig::UP_OR_PK,
    CORBA::Boolean supportIdentityAssertion = static_cast<CORBA::Boolean>(1)
);
```

パラメータ	説明
disable	セキュリティを無効にするかどうかを指定します。
transport	使用する転送メカニズムです。有効な値は次のとおりです。 • csiv2::CLEAR_ONLY：セキュアトランスポートは不要です。 • csiv2::SECURE_ONLY：セキュアトランスポートしか許可されません。 • csiv2::ALL：任意の転送メソッドを使用できます。
trustInClient	転送先がクライアントの認証を要求するかどうかを指定します。この値は，CSIV2 層で設定されます。
access_manager	QoP インプリメンテーションのためのアクセスマネージャで，ユーザーが定義する csiv2::AccessPolicyManager の実装です。null の場合はデフォルト値を使用します。
realms	ポリシーを実装する利用可能な領域です。
requiredIdentity Type	QoP ポリシーのインプリメンテーションに必要な ID です。デフォルト値は，csiv2::ServerQoPConfig::UP_OR_PK です。 指定できる値は，csiv2:ServerQoPConfig::NO_ID，csiv2::ServerQoPConfig::UP，csiv2::ServerQoPConfig::PK，csiv2::ServerQoPConfig::UP_OR_PK，および csiv2::ServerQoPConfig::UP_AND_PK です。
supportIdentity Assertion	アプリケーションが ID アサーションをサポートするかどうかを指定します。

ServerQoPPolicy を定義するには，ポリシーのさまざまな特性を定義するこのオブジェクトを作成します。

インクルードファイル
このクラスを使用するには，CSIV2Policies.h ファイルをインクルードする必要があります。

class ServerQoPPolicyImpl

ServerQoPPolicyImpl は，csiv2::ServerQoPPolicy のインプリメンテーションです。ServerQoPPolicyImpl オブジェクトは，サーバーの QoP の動作に影響します。

インクルードファイル
このクラスを使用するには，CSIV2Policies.h ファイルをインクルードする必要があります。

メソッド

```
ServerQoPPolicyImpl (const csiv2::ServerQoPConfig_var& conf);
```

ServerQoPPolicyImpl オブジェクトのコンストラクタです。

パラメータ	説明
conf	設計された QoP 設定が入っている ServerQoPConfig オブジェクトです。

```
virtual csiv2::ServerQoPConfig_ptr config();
```

ServerQoPPolicyImpl から ServerQoPConfigImpl オブジェクトを取得します。

戻り値 ServerQoPPolicyImpl から ServerQoPConfigImpl オブジェクトが返されます。

class vbsec::ClientConfigImpl

ClientConfigImpl は，csiv2::ClientQoPConfig のインプリメンテーションです。ClientQoPPolicy を定義するには，ポリシーのさまざまな特性を定義するこのオブジェクトを作成します。

インクルードファイル
このクラスを使用するには，CSIV2Policies.h ファイルをインクルードする必要があります。

メソッド

```
ClientConfigImpl (const CORBA::Short transport, const CORBA::Boolean
trustInTarget)
```

ClientConfigImpl オブジェクトのコンストラクタです。

パラメータ	説明
transport	使用する転送メカニズムです。有効な値は次のとおりです。 • csiv2::CLEAR_ONLY：セキュアトランスポートは不要です。 • csiv2::SECURE_ONLY：セキュアトランスポートしか許可されません。 • csiv2::ALL：任意の転送メソッドを使用できます。
trustInTarget	クライアントの認証を必要とするかどうかを指定します。

class vbsec::ClientQoPPolicyImpl

ClientQoPPolicyImpl は，csiv2::ClientQoPPolicy のインプリメンテーションです。ClientQoPPolicyImpl オブジェクトは，サーバーの QoP の動作に影響します。

インクルードファイル

このクラスを使用するには，CSIV2Policies.h ファイルをインクルードする必要があります。

メソッド

```
ClientQoPPolicyImpl( const csiv2::ClientQoPConfig_var& conf);
```

ClientQoPPolicyImpl オブジェクトのコンストラクタです。

パラメータ	説明
conf	ポリシーで使用する ClientConfigImpl オブジェクトです。

```
virtual csiv2::ClientQoPConfig_ptr config();
```

この ClientQopPolicyImpl の ClientConfigImpl オブジェクトを返します。

戻り値 この ClientQopPolicyImpl の ClientConfigImpl オブジェクト。

承認 API

ここでは，VisiSecure での承認に使用するクラスとメソッドについて説明します。

class csiv2::AccessPolicyManager

AccessPolicyManager は，クライアントのメソッド呼び出しの承認のためのアクセスポリシーを定義するために使用します。

インクルードファイル

このクラスを使用するには，CSIV2Policies.h ファイルをインクルードする必要があります。

メソッド

```
char* domain()
```

AccessPolicyManager の承認ドメイン名を返します。

戻り値 この AccessPolicyManager を使用するオブジェクトの承認ドメイン名。

```
csiv2::ObjectAccessPolicy* getAccessPolicy (PortableServer_ServantBase*
servant,
                                            const PortableServer::ObjectId& id,
                                            const CORBA::OctetSequence&
adapter_id)
```

objectId（ID）と poa ID を持ったサーバントの objectAccessPolicy を返します。

パラメータ	説明
servant	CORBA サーバントオブジェクトです。
id	サーバントオブジェクトの ID です。
adapter_id	サーバントオブジェクトの poa ID です。

戻り値 サーバントオブジェクトの ObjectAccessPolicy。

class csiv2::ObjectAccessPolicy

このクラスは，AccessPolicyManager からのアクセスポリシーを表します。

インクルードファイル

このクラスを使用するには，CSIV2Policies.h ファイルをインクルードする必要があります。

メソッド

```
CORBA::StringSequence* getRequiredRoles (const char* method)
```

メソッドにアクセスするために必要なロールのリストを返します。

パラメータ	説明
method	使用するメソッド名です。

戻り値 メソッドにアクセスするために必要なロールのリスト。

```
char* getRunAsRole (const char* method)
```

メソッドの run-as ロールを返します。このメソッドは，このリリースでは使用されません。

パラメータ	説明
method	使用するメソッド名です。

戻り値 メソッドにアクセスするように設定された run-as ロール。

第 12 章

Security SPI for C++

ここでは，VisiSecure for C++ 用に定義された SPI（Service Provider Interface）のクラスについて説明します。SPI のクラスは高度なセキュリティ機能を提供し，ほかのセキュリティプロバイダが独自のセキュリティサービスのインプリメンテーションを VisiSecure に組み込んで Borland Deployment Platform で使用できるようにします。

プラグインメカニズムと SPI

VisiSecure for C++ は，独自のセキュリティインプリメンテーションの一部をプラグインするインターフェースを提供します。ORB が独自のインプリメンテーションを検索できるように，すべてのプラグインは VisiSecure が提供する REGISTER_CLASS マクロを使ってクラスを登録する必要があります。クラスの名前は，登録時に名前空間を含む完全な名前を指定する必要があります。名前空間は外側の名前空間から始まる「.」または「::」で区切られた文字列を使用する VisiSecure がサポートする標準形式で指定する必要があります。次に例を示します。

```
MyNameSpace {
  class MyLoginModule {
    ......
  }
}
```

この場合，MyLoginModule は MyNameSpace.MyLoginModule または MyNameSpace::MyLoginModule と指定します。

プラグイン可能コンポーネントは次の 6 つです。

- **LoginModule**：vbsec::LoginModule を拡張して独自のログインモデルを実装できます。ログインモジュールを使用するには，ほかのログインモジュールと同様に，認証設定ファイルで設定する必要があります。
- **コールバックハンドラ**：vbsec::CallbackHandler を拡張して独自のコールバックを実装できます。コールバックを使用するには，ほかのコールバックハンドラと同様に，認証設定ファイルで設定する必要があります。
- **ID アダプタ，メカニズムアダプタ，および認証メカニズム**：このインターフェースは，ユーザーが独自の認証メカニズムと ID 解釈を実装するために用意されています。IdentityAdaptor は ID を解釈し，MechanismAdaptor は特殊な ID アダプタで，ターゲッ

ト情報も変更します。AuthenticationMechanism はユーザーを認証するプラグイン可能サービスです。

これらのプラグインを使用するには, vbroker.security.identity.xxx プロパティを設定して，プラグインおよびそのプロパティを定義する必要があります。たとえば，ID アダプタまたはメカニズムアダプタは，次のように指定できます。

```
vbroker.security.identity.adapters=MyAdapter
vbroker.security.adapter.MyAdapter.property1=value1
vbroker.security.adapter.MyAdapter.property2=value1
```

また，認証メカニズムは次のとおりです。

```
vbroker.security.identity.mechanisms=MyMechanism
vbroker.security.adapter.MyMechanism.property1=value1
vbroker.security.adapter.MyMechanism.property2=value2
```

指定されたプロパティは，初期化時に文字列マップとしてユーザープラグインに渡されます。マップには property1，value1 のような切り詰められたキーと値の組み合わせが含まれます。

- **属性コーデック：**これにより，属性コーデックをプラグインして，独自の形式で属性をエンコードおよびデコードできます。VisiSecure for C++ には組み込みのコーデックが 1 つ（ATS コーデック）あります。

  コーデックのプラグインを使用するには，プロパティを設定してコーデックおよびそのプロパティを定義する必要があります。次に例を示します。

  ```
  vbroker.security.identity.attributeCodecs=MyCodec
  vbroker.security.adapter.attributeCodec.property1=xxx
  vbroker.security.adapter.attributeCodec.property2=xxx
  ```

  指定されたプロパティは，初期化時に文字列マップとしてユーザープラグインに渡されます。

- **承認サービスプロバイダ：**各承認ドメインに 1 つの承認サービスをプラグインできます。VisiSecure には，ロールマップを使用するデフォルトのインプリメンテーションがあります。ほかのプラグイン可能サービスと同様に，後で文字列マップとして渡されるプロパティを使って承認サービスを定義する必要があります。次に例を示します。

  ```
  vbroker.security.auth.domains=MyDomain
  vbroker.security.domain.MyDomain.provider=MyProvider
  vbroker.security.domain.MyDomain.property1=xxx
  vbroker.security.domain.MyDomain.property2=xxx
  ```

- **信頼プロバイダ：**これにより，アサーション信頼メカニズムをプラグインできます。アサーションは複数のホップシナリオで発生するか, またはアサーション API を介して明示的に呼び出されます。サーバーには，ピアが信頼されてアサーションを生成できるかどうかを決定するルールを設定できます。デフォルトのインプリメンテーションは，プロパティ設定を使ってサーバー側で信頼されたピアを設定します。ピアは実行時に認証と承認を渡し，アサーションを生成するための信頼を確立する必要があります。

  ほかのプラグイン可能サービスと同様に，後で文字列マップとして渡されるプロパティを使って承認サービスを定義する必要があります。次に例を示します。

  ```
  vbroker.security.trust.trustProvider=MyProvider
  vbroker.security.trust.trustProvider.MyProvider.property1=xxx
  vbroker.security.trust.trustProvider.MyProvider.property2=xxx
  ```

  セキュリティサービス全体で指定できる信頼プロバイダは 1 つだけです。

プロバイダ

各プロバイダインスタンスは，VisiSecure が Java リフレクション API を使って作成します。インスタンスが構築されると，実装元が提供する initialize メソッドが呼び出され，個々のインプリメンテーションで使用できるオプションのマップを渡します。オプションのエントリは，個々のプロバイダの実装元が定義します。ユーザーはプロパティファイルでオプションを指定し，VisiSecure はプロパティを解析してオプションを対応するプロバイダに渡します。次の表に，異なるプロバイダのインプリメンテーションをプラグインするためのプロパティを示します。

表 12.1 SPI（Security Service Provider）のインプリメンテーションの設定

モジュール名	設定するプロパティ	実装するインターフェース	オプションのプレフィクス
IdentityAdapter	vbroker.security.identity.adapters	vbsec::IdentityAdapter	vbroker.security.identity.adapter.<name>
AuthenticationMechanism	vbroker.security.identity.mechanisms	vbsec::AuthenticationMechanisms	vbroker.security.identity.mechanism.<name>
AttributeCodec	vbroker.security.identity.attributeCodecs	vbsec::AttributeCodec	vbroker.security.identity.attributeCodec.<name>
TrustProvider	vbroker.security.trustProvider	vbsec::TrustProvider	vbroker.security.trust.trustProvider.<name>

上の表の説明は，次のとおりです。

- 最初の列は，プロバイダモジュールの名前です。
- 2 列目は，各モジュールを定義するために設定するプロパティです。複数のモジュールがある場合は，カンマで区切ります。たとえば，次のプロパティは，ORB のために 2 つの IdentityAdapter インプリメンテーションがインストールされています。

```
vbroker.security.identity.adapters=ID_ADA1,ID_ADA2
```

- 3 列目は，各モジュールが実装する必要があるインターフェースです。インターフェースは，実装元とコア VisiSecure 間の規約を定義します。
- 最後の列は，個々のモジュールのオプションのプレフィクスです。ORB はプロパティファイルを解析し，対応するエントリを初期メソッドの（マップオプション）パラメータとして各モジュールに渡します。たとえば，前の例で定義されている ID_ADA1 IdentityAdapter では，vbroker.security.identity.adapters.ID_ADA1 プレフィクスが付いているすべてのエントリは，ID_ADA1 IdentityAdapter の初期メソッドに渡されます。

プロバイダと例外

初期化中に障害が発生すると，initialize メソッドは InitializationException のインスタンスを生成します。プロバイダの特定のカテゴリでは，複数のインプリメンテーションが共存しているために複数のインスタンスが存在します。それぞれのインプリメンテーションは VisiSecure システム内で initialize メソッドの最初のパラメータとして渡される名前で識別されます。プロバイダの一部のカテゴリでは，ORB 全体に 1 つのインスタンスだけが存在します。たとえば TrustProvider の場合，initialize メソッドにはオプションマップというパラメータだけが存在します。

vbsec::LoginModule

LoginModule はすべてのログインモジュールの親として機能します。ユーザープラグインのログインモジュールは，このクラスを拡張する必要があります。ログインモジュールは，認証設定ファイルで設定され，ログインプロセス時に呼び出されます。ログインモジュールは，指定されたサブジェクトを認証し，対応するプリンシパルと認証情報をサブジェクトに関連付けます。また，ログアウト時に一連のセキュリティ情報を削除および破棄します。

インクルードファイル

このクラスを使用するには，vbauthn.h ファイルをインクルードする必要があります。

メソッド

```
void initialize (Subject* subj=0,
                 CallbackHandler *handler=0,
                 LoginModule::states* sharedStates=0,
                 LoginModule::options* options=0)
```

このメソッドは，ログインモジュールを初期化します。

引数 このメソッドは，次の 4 つの引数を使用します。

- subj : 認証されるサブジェクト。
- handler : 使用するコールバックハンドラ。
- sharedStates : ほかのログインモジュールが提供する追加の認証状態。現在は使用されていない。
- options : 認証設定ファイルで指定する設定オプション。

戻り値 void。

```
bool login()
```

ログインを実行します。これはログインプロセス時に呼び出されます。ログインモジュールはモジュールにあるサブジェクトを認証し，ログインが正常に行われたかどうかを判断します。

戻り値 ログインが成功した場合は true，そうでない場合は false。

```
bool logout()
```

ログアウトを実行します。これはログアウトプロセス時に呼び出されます。ログインモジュールはモジュールにあるサブジェクトをログアウトし，ログアウトが正常に行われたかどうかを判断します。ログインモジュールは，ログイン時に確立された認証情報または ID を削除して破棄します。

戻り値 ログアウトが成功した場合は true，そうでない場合は false。

```
bool commit()
```

ログインをコミットします。これは，ログインプロセスの一部で，ログインが成功した場合に，対応するログインモジュールに指定されている設定オプションに基づいて呼び出されます。ログインモジュールは，独自の認証が成功すると，モジュールのサブジェクトに対応するプリンシパルと認証情報が関連付けられます。そうでない場合，ログインモジュールはそれまでに保存されているすべての状態を削除および破棄します。

戻り値 コミットが成功した場合は true，そうでない場合は false。

```
bool abort()
```

ログインを中止します。これは，ログインプロセスの一部で，ログイン全体が失敗した場合に，ログインモジュールに指定されている設定オプションに基づいて呼び出されます。ログインモジュールはそれまでに保存されているすべての状態を削除および破棄します。

戻り値 中止が成功した場合は true，そうでない場合は false。

vbsec::CallbackHandler

CallbackHandler は，認証情報およびその他の情報の認証に必要なユーザーコールバックを生成するメカニズムです。7 種類のコールバックが用意されています。対話型テキストモードのすべてのコールバックを処理するデフォルトのハンドラがあります。

インクルードファイル

このクラスを使用するには，vbauthn.h ファイルをインクルードする必要があります。

メソッド

```
void handle (Callback::array& callbacks)
```

コールバックを処理します。

引数 処理する callbacks の配列。

戻り値 void。

vbsec::IdentityAdapter

IdentityAdapter は，特定のメカニズムにバインドします。IdentityAdapter の主な目的は，メカニズム固有の ID を解釈することです。IdentityAdapter は，エンティティのメカニズム固有の表現とメカニズム独立の表現の間のエンコードとデコードを実行するために使用します。

VisiSecure に入っている IdentityAdapter

VisiSecure は，次の IdentityAdapter を提供します。

- 「anonymous」という名前の AnonymousAdapter
- 「DN」という名前の DNAdapter
- X509CertificateAdapter（AuthenticationMechanism サブインターフェースのインプリメンテーションとして）
- GSSUPAuthenticationMechanism（AuthenticationMechanism サブインターフェースのインプリメンテーションとして）

メソッド

```
Virtual void initialize (const std::string& name, ::vbsec::InitOptions&) =0;
```

このメソッドは，指定された名前と一連のオプションを使って IdentityAdapter を初期化します。

引数 このメソッドは，次の 2 つの引数を受け取ります。

- IdentityAdapter の name。
- 指定された IdentityAdapter の InitOptions の集合。

例外 初期化に失敗すると，InitializationException が発生します。

```
virtual std::string getName() const=0;
```

これは，IdentityAdapter の名前を返します。

戻り値 IdentityAdapter の名前。

例外 なし。

```
virtual ::CSI::IdentityToken* exportIdentity(::vbsec::Subject&,
::CSI::IdentityToken&) =0;
```

IdentityAdapter の ID を IdentityToken としてエキスポートします。

引数 ID をエキスポートするサブジェクト。

戻り値 IdentityToken データ。

例外 この IdentityAdapter が認識する認証情報がサブジェクトに見つからない場合は，NoCredentialsException が発生します。

```
virtual void importIdentity (::vbsec::Subject&, ::CSI::IdentityToken&) =0;
```

IdentityToken をインポートし，この ID に関連付けられた適切なプリンシパルを呼び出し元サブジェクトに格納します。

引数 ID をインポートするサブジェクト。

例外 この IdentityAdapter が認識する認証情報がサブジェクトに見つからない場合は，NoCredentialsException が発生します。

```
virtual ::vbsec::Privileges* getPrincipal (::vbsec::Subject&anp;) =0;
```

この ID を表すプリンシパルを返します。このメソッドは，EJB とサーブレットのインターフェースに使用します。

引数 プリンシパルサブジェクト。

戻り値 プリンシパルオブジェクト。

例外 なし。

```
virtual ::vbsec::Privileges* getPrivileges (::vbsec::Subect&) =0;
```

引数 ターゲットサブジェクト。

戻り値 この IdentityAdapter が認識するターゲットサブジェクトの権限属性。

例外 なし。

```
virtual ::vbsec::setPrivileges (::vbsec::Privileges*) =0;
```

このメソッドは，ID の権限属性を設定します。

引数 ID に対して設定する権限属性。

例外 なし。

```
virtual void deleteIdentity (::vbsec::Subject&) =0;
```

このメソッドは，指定されたターゲットサブジェクトに関連付けられたプリンシパルと認証情報を削除します。

引数 この IdentityAdapter が認識する削除対象のプリンシパルと認証情報のターゲットサブジェクト。

例外 なし。

vbsec::MechanismAdapter

IdentityAdapter から拡張された MechanismAdapter には，ターゲット情報を変更する機能が追加されています。これは，リモートサイトで使用されているメカニズムがローカルで使用できない場合に便利です。したがって，リモートサイトに送る前にローカル ID を適合させておく必要があります。

VisiSecure の初期インストールには，MechanismAdapter を直接実装するクラスはありませんが，このインターフェースをサポートする 113 ページの「vbsec::AuthenticationMechanisms」サブインターフェースを実装するクラスがいくつかあります。

メソッド

```
virtual const ::CSI::StringOID_var getOid() const =0;
```

メカニズム OID 文字列表現を返します。たとえば，GSSUP メカニズムの文字列表現は oid:2.23.130.1.1.1 です。

戻り値 メカニズムの OID 文字列。

例外 なし。

```
virtual ::vbsec::Target* getTarget (const std::string& realm, const
std::vector<AppConfigurationEntry*>&) =0;
```

領域名と AppConfigurationEntry オブジェクトのリストを受け取り，対応するターゲットを戻します。

引数 このメソッドは，次の 2 つの引数を受け取ります。

- 領域名
- AppConfigurationEntry オブジェクトのリスト。

戻り値 対応するターゲットオブジェクトを返します。

例外 なし。

```
virtual ::vbsec::Target* getTarget (const ::CSI::GSS_NT_ExportedName&) =0;
```

エンコードされたターゲット表現を表すターゲットオブジェクトを返します。

引数 「IETF RFC2743」で定義されている GSS Mechanism-Independent Exported Name の形式でエンコードされたターゲット。

戻り値 ターゲットオブジェクト。

例外 なし。

vbsec::AuthenticationMechanisms

このクラスは，CSIv2 プロトコルに関連する認証メカニズムをサポートするために必要なすべての機能を提供する本格的なメカニズムを表します。

VisiSecure には，それぞれ GSSUP ベースおよび X509 証明書ベースの認証メカニズムのための次のインプリメンテーションが入っています。

- GSSUPAuthenticationMechanism
- X509CertificateAdapter

上位のインターフェースから継承するメソッドに加えて，AuthenticationMechanism には次のメソッドのカテゴリも定義されています。

認証情報に関連するメソッド

これらのメソッドを使用して，認証情報を取得または破棄します。

```
virtual ::vbsec::Subject* acquireCredentials (::vbsec::Target&,
::vbsec::CallbackHandler*) =0;
```

このメソッドは，指定されたターゲットの認証情報を取得します。取得される認証情報は，認証に必要な情報とメカニズムに依存します。

引数 このメソッドは，次の 2 つの引数を受け取ります。

- ターゲットオブジェクト。
- このターゲットの認証情報を取得するためにユーザーとの通信で使用するコールバックハンドラ。

戻り値 取得された認証情報が入っているサブジェクト（操作に失敗すると null になります）。

例外 なし。

```
virtual ::vbsec::Subject* acquireCredentials (const std::string& target,
::vbsec::CallbackHandler*) =0;
```

このメソッドは，指定されたターゲットの文字列表現に対する認証情報を取得します。取得される認証情報は，認証に必要な情報とメカニズムに依存します。

引数 このメソッドは，次の 2 つの引数を受け取ります。

- ターゲットの文字列表現。
- 認証情報を取得するためにユーザーとの通信で使用する対応するコールバックハンドラ。

戻り値 取得された認証情報が入っているサブジェクトオブジェクト（操作に失敗すると null になります）。

例外 なし。

```
virtual void destroyPrivateCredentials (::vbsec::Subject&) =0;
```

このメソッドは，指定されたサブジェクトの秘密認証情報を破棄します。

引数 秘密認証情報を破棄するサブジェクト。

例外 なし。

コンテキストに関連するメソッド

```
virtual ::CORBA::OctetSeq* createInitContext (::vbsec::Subject&) =0;
```

メカニズム固有のクライアント認証トークンを返します。トークンは，指定されたターゲットの認証情報を表します。

引数 ターゲットサブジェクト。

戻り値 指定されたターゲットサブジェクトの認証トークン。

例外 このサブジェクトにこのメカニズムが認識する認証情報が存在しない場合は，NoCredentialsException が発生します。

```
virtual ::vbsec::Target* processInitContext (::vbsec::Subject&,
::CORBA::OctetSeq&) =0;
```

このメソッドは，メカニズム固有のクライアント認証トークンを使用済みにします。初期の認証トークンはデコードされ，メソッドは指定されたサブジェクトに対応する認証情報を格納します。

引数 認証情報を格納するサブジェクト。

例外 なし。

```
virtual ::CSI::GSSToken* createFinalContext (::vbsec::Subject&) =0;
```

このメソッドは，クライアントに返す最終的なコンテキストトークンを作成します。

引数 サブジェクト。

戻り値 最終的なコンテキストトークン。

例外 なし。

```
virtual void processFinalContext (::vbsec::Subject&, ::CORBA::OctetSeq&) =0;
```

サーバーが戻す最終的なコンテキストトークンを使用します。

引数 ターゲットサブジェクト。

例外 なし。

```
virtual ::CSI::GSSToken* createErrorContext (::vbsec::Subject&) =0;
```

認証に失敗した場合に使用するエラーコンテキストトークンを作成します。

引数 ターゲットサブジェクト。

戻り値 エラーコンテキストトークン。

例外 なし。

```
virtual ::vbsec::Subject* processErrorContext (::vbsec::Subject&,
::CSI::GSSToken&, ::vbsec::CallbackHandler*) =0;
```

サーバーが返すエラートークンを使用します。認証情報を再取得するユーザーとの対話にはコールバックハンドラを使用します。認証情報が必要な場合，クライアント側のセキュリティサービスは再びコンテキストを確立しようとします。

引数 このメソッドは，次の 2 つの引数を受け取ります。

- ターゲットサブジェクト。
- コールバックハンドラ。

例外 なし。

vbsec::Target

このクラスは，ターゲットの認証プリンシパルの実行時表現を提供します。コンテキストには，表示名，OID の DER 表現などのさまざまな状況で必要になるターゲットの名前が入っています。

メソッド

```
virtual std::string getName () const =0;
```

このメソッドは，ターゲットの表示名を返します。

戻り値 ターゲットの名前の文字列。

例外 なし。

```
virtual ::CSI::OID getOid () const =0;
```

このメソッドは，ターゲット OID を返します。

戻り値 ターゲットの OID の文字列。

例外 なし。

```
virtual ::CORBA::OctetSeq getEncodedName () const =0;
```

このメソッドは，メカニズム固有のエンコードされたターゲット名を返します。

戻り値 エンコードされたターゲット名。

例外 なし。

vbsec::AuthorizationServicesProvider

この承認サービスの実装元は，特定のリソースへのアクセスを許可されたオブジェクトの集合を提供します。アクセスを決定するときは，必ず AuthorizationServicesProvider が呼び出されます。承認サービスは，承認ドメインの概念に密接に関連付けられています。承認サービスは各承認ドメインのインプリメンテーションに対してインストールされ，その承認ドメインだけに対して動作します。

AuthorizationServicesProvider は，対応する承認ドメインを構築する際に初期化されます。次のプロパティを使用して，AuthorizationServicesProvider のために実装するクラスを設定します。

```
vbroker.security.domain.<domain-name>.provider
```

このプロパティは，Java リフレクションを使って実行時にロードされます。

承認サービスのもう 1 つのインポート機能は，指定された特定のロールの run-as エリアスを返すことです。セキュリティサービスは，エリアスによって識別される一連の ID によって設定されています。リソースが特定のロールでの「run-as」を要求すると，AuthorizationServices が再度呼び出されて，この承認ドメインで指定されているルールのコンテキストで「run-as」を実行するために使用するエリアスを返すように求められます。

メソッド

```
virtual void initialize (const std::string& name, ::vbsec::InitOptions&
options) =0;
```

このメソッドは，承認サービスプロバイダを初期化します。

引数 このメソッドは，次の引数を受け取ります。

- プロバイダ名
- プロバイダのオプション。

プロバイダのオプションに加えて，この承認サービスプロバイダと VisiBroker ORB 間の対話を実現するために次の情報が渡されます。

名前	説明
ORB	現在のシステムで使用される ORB インスタンスです。
Logger	現在のシステムへのログインに使用される SimpleLogger インスタンスです。
LogLevel	セキュリティログレベルを表す整数値です。

例外 承認プロバイダの初期化に失敗すると，InitializationException が発生します。

```
virtual std::string getName() const =0;
```

この承認サービスのインプリメンテーションの名前を返します。

戻り値 承認サービスの名前。

例外 なし。

```
virtual ::vbsec::PermissionCollection* getPermissions (const ::vbsec::Resource*
resource, const ::vbsec::Privileges* callerPrivileges) =0;
```

アクセスを試みるリソース，および指定された特権に対する同種のアクセス許可属性の集合を返します。

引数 このメソッドは，次の 2 つの引数を受け取ります。

- 呼び出し元の特権。
- アクセスを試みるリソースオブジェクト。

戻り値 PermissionCollection オブジェクトは，このサブジェクトのアクセス許可を表します。

例外 なし。

vbsec::Resource

Resource インターフェースは，リソースを包括的に抽象化します。リソースは，リソースに不可欠な CORBA オブジェクトのリモートメソッドやサーブレットなどのアクセスの任意の対象です。

メソッド

```
virtual std::string getName () const =0;
```

アクセスするリソースの文字列表現を返します。

戻り値 リソースの名前。

例外 なし。

vbsec::Privileges

Privileges クラスは，サブジェクトの特権を抽象化します。このクラスは，第 2 章「セキュリティの概要」属性などの承認権限属性のコンテナです。AuthorizationService は，サブジェクトの特権オブジェクトに基づいて特定のリソースにアクセスする許可があるかどうかを決定します。

特権オブジェクトは，PublicCredentials の 1 つとしてサブジェクトの内部に保存されます。同時に，特権はリファレンス元のサブジェクトへのリファレンスを 1 つ保持します。特権には，DN 属性のマップおよびその他の承認属性のマップも含まれます。

Privileges クラスは，javax.security.auth.Destroyable インターフェースを実装します。

コンストラクタ

```
Privileges (const std::string& name, ::vbsec::Subject& subject);
```

このコンストラクタは，指定された名前で特権オブジェクトを作成し，指定されたサブジェクトに関連付けます。

引数 メソッドは，次の 2 つの引数を受け取ります。

- 特権オブジェクトの名前（実際には，関連付けられたサブジェクトの名前です）。
- ターゲットサブジェクト。

例外 なし。

メソッド

```
::vbsec::Subject& getSubject() const ;
```

このメソッドは，特権オブジェクトが表すサブジェクトを返します。

戻り値 ターゲットサブジェクト。

例外 なし。

```
std::string getSubjectName() const;
```

このメソッドは，関連付けられたサブジェクトオブジェクトの名前を返します。

戻り値 ターゲットサブジェクト。

例外 なし。

```
const ::vbsec::ATTRIBUTE_MAP& getAttributes() const ;
```

このメソッドは，ユーザーの属性マップを返します。

戻り値 ユーザーの属性マップ。

例外 なし。

```
void setDBAttributes (const ::vbsec::ATTRIBUTE_MAP& map);
```

このメソッドは，ユーザーの DN 属性を更新します。

引数 新しい DN 属性マップ。

メモ DN 属性マップが設定されると，特権オブジェクトは基底の DN 属性マップを変更不可に設定します。

例外 なし。

```
const ::vbsec::ATTRIBUTE_MAP* getDNAttributes() const;
```

このメソッドは，特権オブジェクトの DN 属性を返します（null の場合もあります）。

戻り値 ユーザーの DN 属性マップ（変更不可）。

例外 なし。

```
bool isDestroyed() const;
```

このメソッドは，特権オブジェクトが破棄されたかどうかをチェックします。

戻り値 true|false

例外 なし。

```
std::string toString() const;
```

このメソッドは，java.lang.Object のデフォルトの toString インプリメンテーションをオーバーライドし，「<subject name> の特権」の情報を返します。

戻り値 各サブジェクト名の特権のリスト。

例外 なし。

vbsec::AttributeCodec

AttributeCodec オブジェクトは，指定されたサブジェクトの特権の属性のエンコードおよびデコードを行います。これにより，クライアントとサーバーは特権の情報をやり取りできます。特権の情報は，承認の意志決定プロセスの基準として使用しますが，AttributeCodec の選択はサーバーが発行する IOR に提示される情報に基づきます。IOR の内部では，サーバーはサポートされているエンコードスキームに関する情報を公開し，クライアントは指定されたエンコードをサポートする AttributeCodec を選択します。

すべての AttributeCodec インプリメンテーションは IdentityService に登録され，これが承認要素プロセスのインポート／エキスポート中に呼び出されます。

メソッド

```
virtual void initialize (const std::string& name, vbsec::InitOptions& options)
=0;
```

このメソッドは，AttributeCodec インプリメンテーションのこのインスタンスを初期化します。1 つの ORB に複数のインプリメンテーションが存在することがあり，それぞれが内部的に提供される名前を使ってアドレス指定されます。

引数 このメソッドは，次の引数を受け取ります。

- AttributeCodec インプリメンテーション名の文字列。
- プロバイダオプション。

プロバイダのオプションに関しては，初期化中に次の追加情報も渡されます。

名前	説明
ORB	現在のシステムで使用される ORB インスタンスです。
Logger	ログ出力のために現在のシステムに使用される SimpleLogger のインスタンスです。
LogLevel	セキュリティログレベルを表す整数値です。

例外 AttributeCodec オブジェクトの初期化に失敗すると，InitializationException が発生します。

```
virtual std::string getName() const =0;
```

このメソッドは，プロバイダインプリメンテーションの名前を返します。

戻り値 プロバイダ名の文字列。

例外 なし。

```
virtual CSIIOP::ServiceConfigurationList* getPrivilegeAuthorities() const =0;
```

このメソッドは，サポートされている特権の証明機関のリストを返します。

戻り値 特権の証明機関のリスト。

例外 なし。

```
4. virtual CSI::AuthorizationElementType getSupportedEncoding() const = 0;
```

このメソッドは，サポートされている AuthorizationElement 型を返します。

戻り値 AuthorizationElement 型。

例外 なし。

```
virtual bool supportsClientDelegation() const =0;
```

このインプリメンテーションが ClientDelegation をサポートするかどうかを返します。

戻り値 true|false

例外 なし。

```
virtual CSI::AuthorizationToken* encode (const
CSIIOP::ServiceConfigurationList& privilege_authorities, vbsec::Privileges&
caller_privileges, vbsec::Privileges& asserter_privileges) =0;
```

このメソッドは，特権を AuthorizationElements としてエンコードします。このメソッドは，指定された呼び出し元とアサート元の権限属性がある場合は，それをエンコードします。サブジェクトおよび呼び出し元とアサート元の権限マップから特権情報を抽出します。

さらに，AttributeCodec のインプリメンテーション（ClientDelegation をサポートする場合）は，この呼び出し元が提示するクライアントのデリゲーション情報に基づいてアサート元が呼び出し元をアサートできるかどうかを確認することもできます。

引数 このメソッドは，次の引数を受け取ります。

- 呼び出し元の特権属性の集合。
- アサート元の特権属性の集合。

戻り値 エンコードされた呼び出し元とアサート元の特権。

例外 アサーションが許可されない場合は，NoDelegationPermissionException が発生します。

```
virtual void decode (const ::CSI::AuthorizationToken& encoded_attributes,
vbsec::Privileges& caller_privileges, vbsec::Privileges& asserter_privileges)
=0;
```

このメソッドは，承認要素をデコードし，対応する特権オブジェクトに格納します。これは encode メソッドの逆のプロセスです。サーバーは，エンコードされた AuthorizationElements のセットを受け取ると，それを解釈するために AttributeCodec に渡します。使用するエンコードメソッドに基づいて，特定の AttributeCodec が理解する属性を使用します。これにより，呼び出し元またはアサート元の特権が更新されるか，または RolePermission が直接サブジェクトの公開認証情報に追加されます。

引数 このメソッドは，次の引数を受け取ります。

- エンコードされた Authorization Elements の集合。
- 呼び出し元の特権の集合。
- アサート元の特権の集合。

戻り値 このメソッドの返り値はありません。処理が成功すると，この AttributeCode オブジェクトは承認要素で使用できる情報に基づいて適宜呼び出し元またはアサート元の特権マップを更新します。

例外 アサーションが承認されない場合は，NoDelegationPermissionException が発生します。

vbsec::Permission

Permission は，リソースにアクセスするための承認情報を表します。すべてのアクセス許可には，実際のインプリメンテーションだけが解釈できる名前が付いています。

インクルードファイル

このクラスを使用するには，vbsecspishared.h ファイルをインクルードする必要があります。

メソッド

```
bool implies (const Permission& p) const
```

アクセス許可が指定された別のアクセス許可を指しているかどうかを評価します。これは，呼び出し元のアクセス許可がリソースへのアクセスに必要なアクセス許可であるかどうかを判定するために承認プロセス時に使用します。呼び出し元のアクセス許可が必要なアクセス許可である場合はアクセスが許可され，そうでない場合は拒否されます。

引数 評価されるアクセス許可 p。

戻り値 アクセス許可が既存のアクセス許可を指す場合は true，そうでない場合は false。

```
bool operator==(const Permission& p) const
```

アクセス許可が指定された別のアクセス許可と等しいかどうかをチェックします。

引数 評価されるアクセス許可 p。

戻り値 アクセス許可が等しい場合は true，それ以外の場合は false。

```
std::string getName () const
```

アクセス許可の名前を取得します。

戻り値 アクセス許可の名前。

```
std::string getActions () const
```

アクセス許可のアクションを文字列として取得します。実際のインプリメンテーションだけが解釈されます。

戻り値 アクセス許可のアクションの文字列表現。

```
std::string toString () const
```

アクセス許可の文字列表現を取得します。

戻り値 アクセス許可の文字列表現です。

vbsec::PermissionCollection

PermissionCollection はアクセス許可のコレクションを表します。

インクルードファイル

このクラスを使用するには，vbsecspishared.h ファイルをインクルードする必要があります。

メソッド

```
bool implies (const Permission& p) const
```

PermissionCollection が指定されたアクセス許可であるかどうかを評価します。

引数 評価されるアクセス許可 p。

戻り値 PermissionCollection が指定されたアクセス許可である場合は true，そうでない場合は false。

vbsec::RolePermission

RolePermission クラスは，VisiSecure システムにおける承認と信頼の基準を提供します。

コンストラクタ

```
RolePermission (const std::string& role)
```

このコンストラクタは，論理ロールを表す RolePermission オブジェクトを作成します。

引数 この RolePermission オブジェクトが表す論理ロールの文字列。

戻り値 RolePermission オブジェクト。

例外 なし。

メソッド

```
virtual bool implies (const Permission& permission) const;
```

このメソッドは，渡されたアクセス許可オブジェクトがこの RolePermission オブジェクトを表すかどうかをチェックします。このチェックは厳密な一致に基づいており，このメソッドは次のすべての条件が満たされた場合にだけ true（表す）を返します。

1 指定されたアクセス許可オブジェクトは，RolePermission のインスタンスです。

2 指定されたアクセス許可オブジェクトの名前は，この RolePermission の名前と同じです。

引数 チェックするアクセス許可オブジェクト。

戻り値 True | False

例外 なし。

```
virtual std::string getActions() const;
```

このメソッドは，この RolePermission に関連付けられているアクションを返します。

戻り値 RolePermission オブジェクトに関連付けられているアクションはないので，常に null を返します。

例外 なし。

vbsec::TrustProvider

リモートピア（サーバーまたはプロセス）が呼び出し元のかわりに動作する際に ID アサーションを行う場合，最終層サーバーがこのアサーションを行うにはピアを信頼する必要があります。これは，信頼されていないクライアントによるアサーションの実行を防止するためです。

中心的なメソッドは isAssertionTrusted で，指定された呼び出し元のサブジェクトおよびアサート元の特権に基づいてアサーションが信頼できるかどうかを判定するために呼び出されます。このメソッドは，クライアントからサーバーに転送された対応する承認要素が使用された後に基底のインプリメンテーションによって呼び出されます。

最終層サーバーが指定されたアサート元のサブジェクトから ID アサーションを受け入れるかどうかを決定するための信頼ルールは，TrustProvider クラスを使って実装します。TrustProvider クラスは，AttributeCodec オブジェクトと特権のインプリメンテーションに密接に関連付けられています。たとえば，次のような意志決定のインプリメンテーションを提供できます。

1 プロキシ承認属性を表すクラスのインプリメンテーションを提供します。

2 AttributeCodec は，必要な論理を実装し，属性を渡し，サーバー側の呼び出し元サブジェクトにインポートします。118 ページの「vbsec::AttributeCodec」で定義される virtual bool supportsClientDelegation() const =0; メソッドに対して true を返すことも必要です。

3 呼び出し元のプロキシ承認属性とアサート元の特権に基づいてメソッドのインプリメンテーションを提供します。

この種類の信頼の評価は，呼び出し元によって提供されるルールに基づくので前方信頼と呼ばれます。後方信頼では，信頼の評価はターゲットのルールに基づきます。後方信頼は VisiSecure のデフォルト設定です。詳細については，第 2 章「セキュリティの概要」を参照してください。

メソッド

```
virtual void initialize (::vbsec::InitOptions&, std::map<std::string,
std::string>&) =0;
```

このメソッドは，TrustProvider を初期化します。各プロセスに対して存在する TrustProvider インプリメンテーションのインスタンスは 1 つだけです。

引数 プロバイダのオプションに関しては，初期化中に次の追加情報も渡されます。

名前	説明
ORB	現在のシステムで使用される ORB インスタンスです。
Logger	ログ出力のために現在のシステムに使用される SimpleLogger のインスタンスです。
LogLevel	セキュリティログレベルを表す整数値です。

例外 TrustProvider の初期化に失敗すると，InitializationException が発生します。

```
virtual bool isAssertionTrusted (const ::vbsec::Subject&, const
::vbsec::Privileges&) =0;
```

このメソッドは，提供された特権を使用したアサート元による呼び出し元のアサーションが信頼されているかどうかを確認します。このインプリメンテーションは，このプロセスの内部信頼ルールを使ってアサーションの有効性を検証します。

引数 このメソッドは，次の 2 つの引数を受け取ります。

- 呼び出し元。
- アサート元の特権の集合。

戻り値 true | false

例外 なし。

vbsec::InitOptions

InitOptions は，初期化プロセスを円滑に行うために初期化の呼び出し時にユーザープラグインに渡されるデータ構造です。

インクルードファイル

このクラスを使用するには，vbsecspishared.h ファイルをインクルードする必要があります。

データメンバー

```
std::map<std::string, std::string>* options
```

解析されるプロパティ設定を表す名前と値の組み合わせを含む文字列マップ。

```
::PortableInterceptor::ORBInitInfo* initInfo
```

ORB 初期化情報を表すオブジェクト。

```
::IOP::Codec* codec
```

IOP Codec オブジェクト。

```
::vbsec::SimpleLogger* logger
```

ロガーオブジェクト。

```
int logLevel
```

セキュリティサービスに現在設定されているログレベル。

vbsec::SimpleLogger

SimpleLogger は，さまざまなレベルの情報をログに記録するメカニズムです。現在は，LEVEL_WARNING，LEVEL_NOTICE，LEVEL_INFO，LEVEL_DEBUG という 4 つのレベル（詳細度が低い方から高い方の順）をサポートします。ロガーはセキュリティサービス全体で 1 つだけです。

インクルードファイル

このクラスを使用するには，vbsecspishared.h ファイルをインクルードする必要があります。

メソッド

```
::std::ostream& WARNING()
```

警告メッセージのログ出力ストリームを返します。

戻り値 LEVEL_WARNING のログ出力ストリーム。

```
::std::ostream& NOTICE()
```

通知メッセージのログ出力ストリームを返します。

戻り値 LEVEL_NOTICE のログ出力ストリーム，またはログレベルが LEVEL_NOTICE より低く設定されている場合は偽ストリーム。

```
::std::ostream& INFO()
```

情報メッセージのログ出力ストリームを返します。

戻り値 LEVEL_INFO のログ出力ストリーム，またはログレベルが LEVEL_INFO より低く設定されている場合は偽ストリーム。

```
::std::ostream& DEBUG()
```

デバッグメッセージのログ出力ストリームを返します。

戻り値 LEVEL_DEBUG のログ出力ストリーム，またはログレベルが LEVEL_DEBUG より低く設定されている場合は偽ストリーム。

索引

記号

数値

A

B

C

D

G

H

T

V

W

X

あ

い

お

か

き

く

け

こ

さ

し

せ

そ

た

て

と

に

は

ひ

ふ

へ

ほ

ま

め

ゆ

ら

り

る

れ

ろ

わ