Oracle® Solaris セキュリティーサービス開発ガイド

Part No: 819–0396–15
2010 年 9 月

ORACLE®

100908@24661

目次

図目次

表目次

例目次

はじめに

『Oracle Solaris セキュリティーサービス開発ガイド』では、Oracle Solaris オペレーティングシステムのセキュリティー機能に関連する公開アプリケーションプログラミングインタフェース (Application Programming Interfaces、API) およびサービスプロバイダインタフェース (Service Provider Interfaces、SPI) について説明します。「サービスプロバイダ」という用語は、暗号化アルゴリズムやセキュリティープロトコルといった、セキュリティーサービスを提供する目的でフレームワークにプラグインされるコンポーネントを指します。

注 – Solaris のこのリリースでは、SPARC および x86 系列のプロセッサアーキテクチャー (UltraSPARC 、SPARC64、AMD64、Pentium、および Xeon EM64T) を使用するシステムをサポートします。サポートされるシステムについては、Solaris 10 Hardware Compatibility List (`http://www.sun.com/bigadmin/hcl`) を参照してください。本書では、プラットフォームにより実装が異なる場合は、それを特記します。

本書の x86 に関連する用語については、以下を参照してください。

- 「x86」は、64 ビットおよび 32 ビットの x86 互換製品系列を指します。
- 「x64」は、AMD64 または EM64T システムに関する 64 ビット特有の情報を指します。
- 「32 ビット x86」は、x86 をベースとするシステムに関する 32 ビット特有の情報を指します。

サポートされるシステムについては、Solaris 10 Hardware Compatibility List を参照してください。

対象読者

『Oracle Solaris セキュリティーサービス開発ガイド』は、次の種類のプログラムを記述する C 言語開発者を対象にしています。

- システム制御を無効化できる特権付きアプリケーション
- 認証と関連セキュリティーサービスを使用するアプリケーション
- ネットワーク通信のセキュリティーを確保する必要のあるアプリケーション

- 暗号化サービスを使用するアプリケーション
- セキュリティーサービスを提供または利用するライブラリ、共有オブジェクト、およびプラグイン

注 – Solaris 機能と同等の Java 言語機能については、`http://java.sun.com/javase/technologies/security/` を参照してください。

お読みになる前に

このマニュアルの読者は、C プログラミングに精通している必要があります。セキュリティー機構の基本的な知識があると便利ですが、必須ではありません。このマニュアルを使用するにあたっては、ネットワークプログラミングについての専門知識は必要ありません。

内容の紹介

このマニュアルは次の章で構成されています。

- 第 1 章「Oracle Solaris の開発者向けセキュリティー機能 (概要)」では、Solaris のセキュリティー機能を紹介します。
- 第 2 章「特権付きアプリケーションの開発」では、プロセス特権を使用する特権付きアプリケーションの記述方法について説明します。
- 第 3 章「PAM アプリケーションおよび PAM サービスの記述」では、Pluggable Application Module (PAM) の記述方法について説明します。
- 第 4 章「GSS-API を使用するアプリケーションの記述」では、Generic Security Service Application Programming Interface(GSS-API) を紹介します。
- 第 5 章「GSS-API クライアント例」と第 6 章「GSS-API サーバー例」では、GSS-API 例について段階的に説明します。
- 第 7 章「SASL を使用するアプリケーションの記述」では、Simple Authentication Security Layer (SASL) アプリケーションの記述方法について説明します。
- 第 8 章「Oracle Solaris 暗号化フレームワークの紹介」では、Solaris 暗号化フレームワークのユーザーレベルとカーネルレベルの両方について概要を説明します。
- 第 9 章「ユーザーレベルの暗号化アプリケーションとプロバイダの記述」では、Solaris 暗号化フレームワークのユーザーレベルのコンシューマおよびプロバイダの記述方法について説明します。
- 第 10 章「スマートカードフレームワークの使用」では、Solaris スマートカードフレームワークについて説明します。

- 付録 A「C ベースの GSS-API プログラム例」では、GSS-API のソースコード例を提供します。
- 付録 B「GSS-API リファレンス」では、GSS-API のさまざまな項目に関するリファレンス情報を提供します。
- 付録 C「OID の指定」では、機構の指定方法について説明します。この技術は、デフォルト以外の機構を使用する必要がある場合に役立ちます。
- 付録 D「SASL ソースコード例」では、SASL 例の完全なソースコードを提供します。
- 付録 E「SASL リファレンス」では、主な SASL インタフェースについて簡単に説明します。
- 付録 F「暗号化プロバイダのパッケージ化と署名」では、暗号化プロバイダをパッケージ化および署名する方法について説明します。
- 用語集では、このマニュアルで使用されているセキュリティー用語の定義を提供します。

関連マニュアル

セキュリティー機能に関するその他の情報については、次のマニュアルを参照してください。

- 『Solaris のシステム管理 (セキュリティサービス)』は、Solaris セキュリティー機能についてシステム管理者の視点から説明しています。
- 『アプリケーションパッケージ開発者ガイド』.
- 『Generic Security Service Application Program Interface』 (ftp://ftp.isi.edu/in-notes/rfc2743.txt) は、GSS-API の概念について簡単に説明しています。
- 『Generic Security Service API Version 2: C-Bindings』 (ftp://ftp.isi.edu/in-notes/rfc2744.txt) は、C 言語ベースの GSS-API に固有の情報を提供しています。
- 『ONC+ 開発ガイド』は、リモートプロシージャー呼び出しに関する情報を提供しています。

マニュアル、サポート、およびトレーニング

追加リソースについては、次の Web サイトを参照してください。

- マニュアル (`http://docs.sun.com`)
- サポート (`http://www.oracle.com/us/support/systems/index.html`)
- トレーニング (`http://education.oracle.com`) – 左のナビゲーションバーで「Sun」のリンクをクリックします。

Oracle へのご意見

Oracle はドキュメントの品質向上のために、お客様のご意見やご提案をお待ちしています。誤りを見つけたり、改善に向けた提案などがある場合は、http://docs.sun.com で「Feedback」をクリックしてください。可能な場合には、ドキュメントのタイトルやパート番号に加えて、章、節、およびページ番号を含めてください。返信を希望するかどうかもお知らせください。

Oracle Technology Network (http://www.oracle.com/technetwork/index.html) では、Oracle ソフトウェアに関する広範なリソースが提供されています。

- ディスカッションフォーラム (http://forums.oracle.com) で技術的な問題や解決策を話し合う。
- Oracle By Example (http://www.oracle.com/technology/obe/start/index.html) のチュートリアルで、手順に従って操作を体験する。
- サンプルコード (http://www.oracle.com/technology/sample_code/index.html) をダウンロードする。

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 P–1　表記上の規則

字体または記号	意味	例
AaBbCc123	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 system%
AaBbCc123	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して示します。	system% **su** password:
AaBbCc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、rm *filename* と入力します。
『 』	参照する書名を示します。	『コードマネージャ・ユーザーズガイド』を参照してください。
「 」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。

表 P–1 表記上の規則 (続き)

字体または記号	意味	例
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	`sun% grep '^#define \` `XV_VERSION_STRING'`

Oracle Solaris OS に含まれるシェルで使用する、UNIX のデフォルトのシステムプロンプトとスーパーユーザープロンプトを次に示します。コマンド例に示されるデフォルトのシステムプロンプトは、Oracle Solaris のリリースによって異なります。

- C シェル

  ```
  machine_name% command y|n [filename]
  ```

- C シェルのスーパーユーザー

  ```
  machine_name# command y|n [filename]
  ```

- Bash シェル、Korn シェル、および Bourne シェル

  ```
  $ command y|n [filename]
  ```

- Bash シェル、Korn シェル、および Bourne シェルのスーパーユーザー

  ```
  # command y|n [filename]
  ```

[ ] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: Shift キーを押します)。ただし、キーボードによっては Enter キーが Return キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、Ctrl-D は Control キーを押したまま D キーを押すことを意味します。

◆ ◆ ◆ 第 1 章

1 Oracle Solaris の開発者向けセキュリティー機能 (概要)

このマニュアルでは、Solaris オペレーティングシステム (Solaris OS) のセキュリティー機能に対する公開アプリケーションプログラミングインタフェース (API) と公開サービスプロバイダインタフェース (SPI) について説明します。

この章の内容は次のとおりです。

Oracle Solaris の開発者向けセキュリティー機能の概要

このマニュアルでは、Solaris オペレーティングシステムのセキュリティー機能に対する公開 API と公開 SPI について説明します。システム管理者の視点から見たこれらのセキュリティー機能の動作については、『Solaris のシステム管理 (セキュリティサービス)』の第 1 章「セキュリティーサービス (概要)」を参照してください。

Solaris OS は、業界標準のインタフェースに基づくネットワークセキュリティーアーキテクチャーを提供します。標準化されたインタフェースを使用すれば、暗号化サービスを使用または提供するアプリケーションを、セキュリティー技術の進歩に合わせて変更する必要がなくなります。

システムセキュリティー

システムのセキュリティーに関して、Solaris OS はプロセス特権を提供します。「プロセス特権」は、特権付きアプリケーションへのアクセス許可を行う際に、スーパーユーザーベースの UNIX 標準モデルに対する代替技術として使用できます。システム管理者は、特権付きアプリケーションへのアクセスを許可する一連のプロセス特権をユーザーに付与します。そうしたユーザーは、スーパーユーザーにならなくても、特権付きアプリケーションを使用できます。

特権を使用すれば、システム管理者は、システムセキュリティーを無視する権限をユーザーに与える際に、完全なスーパーユーザー権限ではなく制限された権限を与えることができます。したがって、新しい特権付きアプリケーションを作成する開発者は、UID が 0 かどうかを検査するのではなく、特定の特権の有無を検査する必要があります。第 2 章「特権付きアプリケーションの開発」を参照してください。

きわめて強固なシステムセキュリティーが必要な場合には、Oracle Solaris の Trusted Extensions 機能の使用を検討してください。Trusted Extensions 機能を使用すれば、システム管理者は、特定のユーザーだけがアクセスできるアプリケーションやファイルを指定できます。Trusted Extensions 機能はこのマニュアルの対象外です。詳細については、`http://www.sun.com/software/solaris/ds/trusted_extensions.jsp` を参照してください。

ネットワークセキュリティーアーキテクチャー

ネットワークセキュリティーアーキテクチャーは、PAM、GSS-API、SASL、RSA Security Inc. PKCS#11 Cryptographic Token Interface (Cryptoki) といった業界標準のインタフェースに対応しています。標準化されたプロトコルやインタフェースを使用することで、開発者は、セキュリティー技術が進歩しても変更する必要のないコンシューマとプロバイダを記述できます。

セキュリティーサービスを使用するアプリケーション、ライブラリ、またはカーネルモジュールは「コンシューマ」と呼ばれます。コンシューマにセキュリティーサービスを提供するアプリケーションは、「プロバイダ」または「プラグイン」と呼ばれます。暗号化処理を実装するソフトウェアは「機構」と呼ばれます。機構は単なるアルゴリズムではなく、アルゴリズムの適用方法も含んだ概念です。たとえば、ある機構は DES アルゴリズムを認証に適用します。一方、別の機構は DES をデータ保護 (ブロック単位の暗号化) に適用します。

ネットワークセキュリティーアーキテクチャーを使えば、コンシューマ開発者が暗号化アルゴリズムを記述、保守、および最適化する必要がなくなります。最適化された暗号化機構が、アーキテクチャーの一部として提供されます。

Solaris OS が提供する公開 Solaris セキュリティーインタフェースは、次のとおりです。

- **PAM** – プラグイン可能な認証モジュール (Pluggable Authentication Module)。PAM モジュールは主に、初期段階でシステムに対してユーザーを認証する際に使用されます。ユーザーはログイン時に、GUI、コマンド行、またはその他の方法を使用できます。PAM は、認証サービスのほかに、アカウント、セッション、およびパスワードを管理するためのサービスも提供します。`login`、`rlogin`、`telnet` などのアプリケーションは、PAM サービスの一般的なコンシューマです。PAM SPI は、Kerberos v5 やスマートカードといったセキュリティープロバイダからサービスを提供されます。第 3 章「PAM アプリケーションおよび PAM サービスの記述」を参照してください。
- **GSS-API** – 汎用セキュリティーサービスアプリケーションプログラムインタフェース (Generic Security Service Application Program Interface)。GSS-API は、ピアとなるアプリケーション間のセキュリティー保護された通信を可能にします。また、GSS-API は認証、整合性、機密性の各保護サービスも提供します。GSS-API の Solaris 実装は、Kerberos v5、SPNEGO、および Diffie-Hellman 暗号化に対応しています。GSS-API は主に、セキュリティー保護されたアプリケーションプロトコルを設計または実装する際に使用されます。GSS-API は、SASL など、ほかの種類のプロトコルに対してサービスを提供できます。GSS-API は、SASL 経由で LDAP にサービスを提供します。

  一般に、GSS-API は、初期の資格確立後にネットワーク上で通信している 2 つのピアとなるアプリケーションによって使用されます。特に GSS-API は、ログインアプリケーション、NFS、および ftp によって使用されます。

  GSS-API の概要については、第 4 章「GSS-API を使用するアプリケーションの記述」を参照してください。第 5 章「GSS-API クライアント例」と第 6 章「GSS-API サーバー例」では、2 つの一般的な GSS-API アプリケーションのソースコードについて説明します。付録 A「C ベースの GSS-API プログラム例」では、GSS-API のソースコード例を提供します。付録 B「GSS-API リファレンス」では、GSS-API のリファレンス情報を提供します。付録 C「OID の指定」では、デフォルト以外の機構を指定する方法を説明します。
- **SASL** – 簡易認証セキュリティー層 (Simple Authentication and Security Layer)。SASL は主に、認証、機密性、データ整合性を必要とするプロトコルによって使用されます。SASL は、セキュリティー機構の動的折衝を使用してセッションを保護する、比較的高レベルのネットワークベースアプリケーション向けに設計されています。LDAP は著名な SASL コンシューマの 1 つです。SASL は GSS-API に似ていますが、SASL のほうが GSS-API よりもいくぶん高レベルです。SASL は GSS-API サービスを使用します。第 7 章「SASL を使用するアプリケーションの記述」を参照してください。
- 暗号化フレームワーク – 暗号化フレームワークは、Solaris OS の暗号化サービスの土台です。このフレームワークは、暗号化サービスのコンシューマとプロバイダに対し、標準の PKCS #11 インタフェースを提供します。このフレームワークは 2 つの部分から構成されています。 1 つはユーザーレベルアプリケーション用の

ユーザー暗号化フレームワーク、もう 1 つはカーネルレベルモジュール用のカーネル暗号化フレームワークです。フレームワークに接続するコンシューマは、インストールされている暗号化機構に関する特別な知識を要求されません。フレームワークにプラグインするプロバイダは、さまざまな種類のコンシューマが要求する特別なコードを用意する必要がありません。

暗号化フレームワークのコンシューマとしては、セキュリティープロトコル、特定の機構、暗号化を必要とするアプリケーションなどが挙げられます。このフレームワークのプロバイダとなるのは、ハードウェアプラグインやソフトウェアプラグイン内の暗号化機構やその他の機構です。暗号化フレームワークの概要については、第 8 章「Oracle Solaris 暗号化フレームワークの紹介」を参照してください。このフレームワークのサービスを使用するユーザーレベルアプリケーションの記述方法を学ぶには、第 9 章「ユーザーレベルの暗号化アプリケーションとプロバイダの記述」を参照してください。

暗号化フレームワークのライブラリは、RSA PKCS#11 v2.11 仕様を実装したものです。コンシューマとプロバイダはどちらも、標準 PKCS #11 呼び出しを使ってユーザーレベル暗号化フレームワークと通信します。

- スマートカード – スマートカード端末用 IFD ハンドラの開発者は、スマートカードフレームワークの端末インタフェース経由で、コンシューマにサービスを提供できます。これらのインタフェースについては、第 10 章「スマートカードフレームワークの使用」を参照してください。
- **Java API** – Java セキュリティーテクノロジには多数の API やツールのほか、一般的に使用されるセキュリティーアルゴリズム、機構、およびプロトコルの実装が含まれています。Java セキュリティー API は、暗号化や公開鍵インフラストラクチャー、セキュリティー保護された通信、認証、アクセス制御など、広範な領域をカバーします。Java セキュリティーテクノロジは、アプリケーションを記述するための包括的なセキュリティーフレームワークを開発者に提供するとともに、アプリケーションを安全に管理するためのツール群をユーザーや管理者に提供します。`http://java.sun.com/javase/technologies/security/` を参照してください。

◆ ◆ ◆ 第 2 章

2 特権付きアプリケーションの開発

この章では、特権付きアプリケーションを開発する方法について説明します。この章の内容は次のとおりです。

特権付きアプリケーション

「特権付きアプリケーション」とは、システム制御を無視し、特定のユーザー ID (UID)、グループ ID (GID)、承認、および特権の有無を検査できるアプリケーションのことです。これらのアクセス制御要素はシステム管理者によって割り当てられます。管理者がこれらのアクセス制御要素を使用する方法に関する一般的な説明については、『Solaris のシステム管理 (セキュリティサービス)』の第 8 章「役割と特権の使用 (概要)」を参照してください。

Solaris OS は、きめ細かい特権委託を可能にするために、次の 2 つの要素を開発者に提供します。

- 特権 - 「特権」とは、特定のアプリケーションに付与することのできる個別の権利のことです。特定の特権を持つプロセスは、通常であれば Solaris OS によって禁止されるような操作を実行できます。たとえば、プロセスは通常、適切なファイルアクセス権を持たないデータファイルを開けません。`file_dac_read` 特権は、ファイル読み取り用の UNIX ファイルアクセス権を無視する権限を、プロセスに対して提供します。特権はカーネルレベルで適用されます。
- 承認 - 「承認」とは、通常であればセキュリティーポリシーによって禁止されるような一連のアクションを実行する権限のことです。承認は役割またはユーザーに割り当てることができます。承認はユーザーレベルで適用されます。

承認と特権の違いは、「だれが何を行えるか」というポリシーの適用レベルにあります。特権はカーネルレベルで適用されます。適切な特権を持たないプロセスは、特権付きアプリケーションで特定の操作を実行できません。承認は、ユーザーアプリケーションレベルでポリシーを適用します。承認は、特権付きアプリケーションにアクセスしたり、特権付きアプリケーションで特定の操作を実行したりする際に必要になる可能性があります。

特権について

特権は、通常であればSolaris OSによって禁止されるような操作を実行できるように、特定のプロセスに付与される個別の権利です。大部分のプログラムは特権を使用しません。というのも、プログラムは一般に、システムのセキュリティーポリシーの境界の内側で動作するからです。

特権は管理者によって割り当てられます。特権は、プログラムの設計に従って有効化されます。ログイン時またはプロファイルシェル起動時には、シェル内で実行されるすべてのコマンドに対して、管理者による特権割り当てが適用されます。アプリケーション実行時には、特権のオン/オフがプログラム的に切り替えられます。exec(1) コマンドを使用して新しいプログラムが起動されると、そのプログラムは親プロセスの継承可能な特権のすべてを使用できる可能性があります。ただし、そのプログラムは新しい特権を1つも追加できません。

管理者が特権を割り当てる方法

コマンドに特権を割り当てるのは、システム管理者の責任です。特権の割り当てについての詳細は、『Solarisのシステム管理(セキュリティサービス)』の「特権(概要)」を参照してください。

特権の実装方法

すべてのプロセスは次の4つの特権セットを備えており、これらによって、特定の特権を使用できるかどうかが決まります。

- 許可された特権セット
- 継承可能な特権セット
- 制限特権セット
- 実効特権セット

許可された特権セット

許可されたセットには、プロセスが使用できる可能性のあるすべての特権を含める必要があります。逆に言えば、使用すべきでない特権は、そのプログラムの許可されたセットに含めてはいけません。

プロセスが起動された場合、そのプロセスは親プロセスから許可された特権セットを継承します。一般に、ログイン時や新しいプロファイルシェルが起動される際、許可された特権の初期セットにはすべての特権が含まれます。このセット内の特権は管理者によって指定されます。子プロセスはそれぞれ、許可されたセットから特権を削除することはできますが、許可されたセットにほかの特権を追加することはできません。セキュリティー上、プログラムが決して使用しない特権は、許可されたセットから削除しておく必要があります。そうすることで、誤って割り当てた、または誤って継承された特権をプログラムが使用する心配がなくなります。

許可された特権セットから削除された特権は、実効セットからも自動的に削除されます。

継承可能な特権セット

ログイン時や新しいプロファイルシェルが起動される際、管理者によって指定された特権が継承可能なセットに含められます。これらの継承可能な特権は、exec(1) の呼び出し後に子プロセスに渡される可能性があります。プロセスは不要な特権を削除し、それらの特権が子プロセスに渡されるのを防止すべきです。たいていは、許可されたセットと継承可能なセットの内容は同じになります。ただし、継承可能なセットから削除された特権が許可されたセット内に残される場合もあります。

制限特権セット

制限セットを使えば、開発者は、プロセスが行使したり子プロセスに渡したりできる特権を制御できます。子プロセスや子孫プロセスが取得できるのは、制限セット内に含まれる特権だけです。`setuid(0)` 関数を実行する場合、そのアプリケーションが使用できる特権は、制限セットによって決まります。制限セットは exec(1) の実行時に適用されます。制限セットから特権を削除しても、exec(1) が実行されるまでは、ほかのセットには影響はありません。

実効特権セット

プロセスが実際に使用できる特権が、プロセスの実効セット内に収められます。プログラム起動時の実効セットは、許可されたセットと等しくなります。その後、実効セットは、許可されたセットと等しいか、あるいはそのサブセットになります。

実効セットを基本特権セットに制限することをお勧めします。中核特権を含む基本特権セットは、28 ページの「特権の種類」に記述されています。プログラム内で使用しない特権をすべて削除します。必要になるまですべての基本特権をオフにしておきます。たとえば、`file_dac_read` 特権があれば、すべてのファイルを読み取れます。プログラム内にファイル読み取りルーチンが複数含まれる可能性があります。そうしたプログラムでは、最初にすべての特権をオフにしますが、適切な読み取りルーチンに対して `file_dac_read` をオンにします。したがって、プログラムが間違った読み取りルーチンに対して `file_dac_read` 特権を行使することはありえません。こうした方法は「特権の囲い込み」と呼ばれます。特権の囲い込みの例については、32 ページの「特権のコーディング例」を参照してください。

スーパーユーザーモデルと特権モデルの互換性

従来のアプリケーションにも対応できるように、特権の実装は、スーパーユーザーモデルと特権モデルのどちらでも動作します。この対応は、PRIV_AWARE フラグを使用することで実現されます。このフラグは、プログラムが特権に対応していることを示します。PRIV_AWARE フラグは、オペレーティングシステムによって自動的に処理されます。

特権を認識しない子プロセスについて検討してください。このプロセスの PRIV_AWARE フラグは false になっています。親プロセスから継承された特権のすべてが、許可されたセットと実効セットに含まれます。子プロセスが UID を 0 に設定した場合、その実効セットと許可されたセットは、制限セット内の特権に制限されます。子プロセスはスーパーユーザーの力を完全には行使できません。したがって、特権に対応したプロセスの制限セットが、特権に対応していないすべての子プロセスのスーパーユーザー特権を制限することになります。子プロセスが特権セットをいずれか 1 つでも変更すると、PRIV_AWARE フラグが true に設定されます。

特権の種類

特権は、その適用範囲に基づいて論理的に次のように分類されます。

- 基本特権 – 最小限の操作に必要とされる中核特権。基本特権は次のとおりです。
  - `PRIV_FILE_LINK_ANY` – プロセスが自身の実効 UID 以外の UID が所有するファイルへのハードリンクを作成できるようにします。
  - `PRIV_PROC_EXEC` – プロセスが `execve()` を呼び出せるようにします。
  - `PRIV_PROC_FORK` – プロセスが `fork()`、`fork1()`、または `vfork()` を呼び出せるようにします。
  - `PRIV_PROC_SESSION` – プロセスがシグナルを送信したり、セッション外のプロセスを追跡したりできるようにします。
  - `PRIV_PROC_INFO` – プロセスが、照会プロセスがシグナルを送信できるプロセス以外のプロセスの状況を調べられるようにします。この特権がない場合、`/proc` の下に見ることのできないプロセスは調べることはできません。

  一般に、基本特権は単独ではなくセットとして割り当てる必要があります。そうすれば、Solaris OS のアップデートでリリースされた基本特権のすべてが、割り当てに含まれることが保証されます。これに対し、プログラムで使用されないことがわかっている特権は、明示的にオフにする必要があります。たとえば、exec(1) サブプロセスを実行しないプログラムでは、`proc_exec` 特権をオフにするべきです。
- ファイルシステム特権。
- System V プロセス間通信 (IPC) 特権。

- ネットワーク特権。
- プロセス特権。
- システム特権。

Solaris 特権と説明の完全な一覧については、privileges(5) のマニュアルページを参照してください。

注 – Solaris は、ゾーン機能を提供します。この機能を使用して管理者はアプリケーション実行用の隔離された環境を設定できます。zones(5) のマニュアルページを参照してください。そのゾーン外のシステムのほかの動作はゾーン内のプロセスを監視したり干渉したりできないため、そのプロセスに対するすべての特権もゾーンに限定されます。ただし、必要な場合は、大域ゾーン以外で操作する特権が必要な大域ゾーン内のプロセスに PRIV_PROC_ZONE 特権を適用できます。

特権を使用したプログラミング

ここでは、特権を操作するためのインタフェースについて説明します。特権プログラミングインタフェースを使用するには、次のヘッダーファイルが必要になります。

```
#include <priv.h>
```

また、特権付きアプリケーションにおける特権インタフェースの使用例も示します。

特権のデータ型

特権インタフェースで使用される主なデータ型は、次のとおりです。

- 特権型 – 個々の特権は、priv_t 型定義によって表現されます。priv_t 型の変数を特権 ID 文字列で初期化するには、次のようにします。

  ```
  priv_t priv_id = PRIV_FILE_DAC_WRITE;
  ```

- 特権セット型 – 特権セットは、priv_set_t データ構造体によって表現されます。priv_set_t 型の変数を初期化するには、表 2–1 の特権操作関数のいずれかを使用します。
- 特権操作型 – ファイルまたはプロセスの特権セット上で実行される操作の種類は priv_op_t 型定義によって表現されます。すべての種類の特権セットですべての操作が有効とは限りません。詳細は、29 ページの「特権を使用したプログラミング」の特権セットの説明を参照してください。

  特権操作として指定可能な値は、次のとおりです。

- PRIV_ON – priv_set_t 構造体で表明された特権を、指定されたファイルまたはプロセスの特権セット内でオンにします。
- PRIV_OFF – priv_set_t 構造体で表明された特権を、指定されたファイルまたはプロセスの特権セット内でオフにします。
- PRIV_SET – 指定されたファイルまたはプロセスの特権セット内の特権を、priv_set_t 構造体で表明された特権に設定します。この構造体が空に初期化されていた場合、PRIV_SET は特権セットを none に設定します。

特権インタフェース

次の表は、特権を使用するためのインタフェースの一覧です。この表に続いて、主な特権インタフェースのいくつかについて説明します。

表 2–1 特権を使用するためのインタフェース

目的	関数	補足説明
特権セットの取得と設定	setppriv(2), getppriv(2), priv_set(3C), priv_ineffect(3C)	setppriv() と getppriv() はシステム呼び出しです。priv_ineffect() と priv_set() は簡易操作用のラッパーです。
特権の特定と変換	priv_str_to_set(3C), priv_set_to_str(3C), priv_getbyname(3C), priv_getbynum(3C), priv_getsetbyname(3C), priv_getsetbynum(3C)	これらの関数は、指定された特権または特権セットを特定の名前または数字にマッピングします。
特権セットの操作	priv_allocset(3C), priv_freeset(3C), priv_emptyset(3C), priv_fillset(3C), priv_isemptyset(3C), priv_isfullset(3C), priv_isequalset(3C), priv_issubset(3C), priv_intersect(3C), priv_union(3C), priv_inverse(3C), priv_addset(3C), priv_copyset(3C), priv_delset(3C), priv_ismember(3C)	これらの関数は、特権のメモリー割り当て、テスト、および各種セット操作に関する機能を提供します。

表 2–1 特権を使用するためのインタフェース (続き)

目的	関数	補足説明
プロセスフラグの取得と設定	getpflags(2), setpflags(2)	PRIV_AWARE プロセスフラグは、プロセスが特権を理解するのか、それともプロセスがスーパーユーザーモデルの下で実行されるのかを示します。PRIV_DEBUG は特権のデバッグ時に使用されます。
低レベルの資格操作	ucred_get(3C)	これらのルーチンは、デバッグ、低レベルのシステム呼び出し、およびカーネル呼び出しを行う際に使用されます。

setppriv():特権設定用

特権設定用の主要関数は、 setppriv() です。その構文は次のとおりです。

```
int setppriv(priv_op_t op, priv_ptype_t which, \
const priv_set_t *set);
```

op は、実行する特権操作を表します。*op* パラメータには、次の 3 つの値のいずれかを指定できます。

- PRIV_ON – *set* 変数によって指定された特権を、*which* によって指定されたセット型に追加します
- PRIV_OFF – *set* 変数によって指定された特権を、*which* によって指定されたセット型から削除します
- PRIV_SET – *set* 変数によって指定された特権で、*which* によって指定されたセット型に含まれる特権を置き換えます

which には、変更する特権セットの種類を指定します。次のいずれかを指定します。

- PRIV_PERMITTED
- PRIV_EFFECTIVE
- PRIV_INHERITABLE
- PRIV_LIMIT

set には、変更操作で使用される特権を指定します。

さらに、簡易関数 priv_set() が提供されています。

priv_str_to_set():特権マッピング用

これらの関数は、特権名を数値にマッピングする場合に役立ちます。
priv_str_to_set() は、このファミリの一般的な関数です。priv_str_to_set() の構文は次のとおりです。

```
priv_set_t *priv_str_to_set(const char *buf, const char *set, \
const char **endptr);
```

`priv_str_to_set()` は特権名の文字列を引数に取りますが、その文字列は `buf` に指定されます。`priv_str_to_set()` が返す特権値のセットは、4 つの特権セットのいずれかと組み合わせることができます。`**endptr` は、構文解析エラーのデバッグ時に使用できます。`buf` には次のキーワードを含めることができます。

- 「`all`」は、定義済みのすべての特権を示します。「`all,!`*priv_name*`,...`」の書式を使えば、指定された特権を除くすべての特権を指定できます。

  注 – 「*priv_set*`,"!`*priv_name*`,...`」を使用する構文は指定された特権のセットから指定された特権を取り去ります。最初にセットを指定せずに「`!`*priv_name*`,...`」を使用しないでください。元になる特権セットが指定されていないと、構文は空の特権セットから指定された特権を取り去ることになり、事実上、特権がないことになってしまいます。

- 「`none`」は、特権なしを示します。
- 「`basic`」は、標準 UNIX オペレーティングシステムへのログイン時にすべてのユーザーに従来から許可されている操作を実行する際に必要となる特権のセットを示します。

特権のコーディング例

ここでは、スーパーユーザーモデルを使って特権を囲い込みする方法と、最小特権モデルを使って特権を囲い込みする方法を比較します。

スーパーユーザーモデルでの特権の囲い込み

次の例では、スーパーユーザーモデルで特権操作を囲い込みする方法を示します。

例 2–1 スーパーユーザー特権の囲い込み例

```
/* Program start */
uid = getuid();
seteuid(uid);

/* Privilege bracketing */
seteuid(0);
/* Code requiring superuser capability */
...
/* End of code requiring superuser capability */
seteuid(uid);
...
/* Give up superuser ability permanently */
setreuid(uid,uid);
```

最小特権モデルでの特権の囲い込み

この例では、最小特権モデルで特権操作を囲い込みする方法を示します。この例では、次のように仮定します。

- プログラムは `setuid 0` である。
- `setuid 0` の結果として、許可されたセットと実効セットは最初、すべての特権に設定されている。
- 継承可能なセットは最初、基本特権に設定されている。
- 制限セットは最初、すべての特権に設定されている。

コードリストに続いて、この例の説明があります。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

例 2–2 最小特権の囲い込み例

```
1  #include <priv.h>
2  /* Always use the basic set. The Basic set might grow in future
3   * releases and potentially retrict actions that are currently
4   * unrestricted */
5  priv_set_t *temp = priv_str_to_set("basic", ",", NULL);

6  /* PRIV_FILE_DAC_READ is needed in this example */
7  (void) priv_addset(temp, PRIV_FILE_DAC_READ);

8  /* PRIV_PROC_EXEC is no longer needed after program starts */
9  (void) priv_delset(temp, PRIV_PROC_EXEC);

10 /* Compute the set of privileges that are never needed */
11  priv_inverse(temp);

12  /* Remove the set of unneeded privs from Permitted (and by
13   * implication from Effective) */
14  (void) setppriv(PRIV_OFF, PRIV_PERMITTED, temp);

15  /* Remove unneeded priv set from Limit to be safe */
16  (void) setppriv(PRIV_OFF, PRIV_LIMIT, temp);

17  /* Done with temp */
18  priv_freeset(temp);

19  /* Now get rid of the euid that brought us extra privs */
20  (void) seteuid(getuid());

21  /* Toggle PRIV_FILE_DAC_READ off while it is unneeded */
22  priv_set(PRIV_OFF, PRIV_EFFECTIVE, PRIV_FILE_DAC_READ, NULL);

23  /* Toggle PRIV_FILE_DAC_READ on when special privilege is needed*/
24  priv_set(PRIV_ON, PRIV_EFFECTIVE, PRIV_FILE_DAC_READ, NULL);
```

例 2–2 最小特権の囲い込み例 *(続き)*

```
25  fd = open("/some/retricted/file", O_RDONLY);

26  /* Toggle PRIV_FILE_DAC_READ off after it has been used */
27  priv_set(PRIV_OFF, PRIV_EFFECTIVE, PRIV_FILE_DAC_READ, NULL);

28  /* Remove PRIV_FILE_DAC_READ when it is no longer needed */
29  priv_set(PRIV_OFF, PRIV_ALLSETS, PRIV_FILE_DAC_READ, NULL);
```

このプログラムでは、*temp* という名前の変数が定義されます。*temp* 変数は、このプログラムで不要な特権のセットを判定します。まず 5 行目で、*temp* が基本特権セットを含むように定義されています。7 行目で、`file_dac_read` 特権が *temp* に追加されています。`proc_exec` 特権は、新しいプロセスに対して exec(1) を実行するために必要ですが、このプログラムでは許可されていません。したがって、9 行目で、`proc_exec` が *temp* から削除されています。これで、exec(1) コマンドを使って新しいプロセスを実行できなくなります。

この時点で *temp* に含まれているのは、このプログラムが必要とする特権だけ、つまり基本セット + `file_dac_read` - `proc_exec` です。11 行目では、`priv_inverse()` 関数が、*temp* の反転を計算し、*temp* の値をその反転値にリセットしています。この反転は、指定されたセット (この場合は *temp*) を可能なすべての特権のセットから差し引いた結果です。11 行目を実行した結果、*temp* には、このプログラムが必要としない特権が含まれています。14 行目で、*temp* によって定義された不要な特権が、許可されたセットから差し引かれています。この削除の結果、それらの特権が実効セットからも削除されます。16 行目で、不要な特権が制限セットから削除されています。18 行目で、*temp* 変数は解放されています。というのも、*temp* は以後使用しないからです。

このプログラムは特権に対応しています。したがって、このプログラムでは `setuid` は使用されず、20 行目で実効 UID がユーザーの実際の UID にリセットされています。

22 行目では、`file_dac_read` 特権を実効セットから削除することで、この特権が無効化されています。実際のプログラムでは、`file_dac_read` が必要とされる前に、何らかの処理が行われます。このサンプルプログラムでは、25 行目でファイルを読み取る際に `file_dac_read` が必要となります。 したがって、24 行目で `file_dac_read` が有効化されています。 ファイルの読み取り後すぐに、`file_dac_read` が実効セットから再度削除されています。すべてのファイルの読み取りが完了すると、すべての特権セット内の `file_dac_read` をオフにすることで、`file_dac_read` が完全に削除されています。

次の表は、プログラム実行中の特権セットの遷移を示したものです。行番号が表示されています。

表2–2 特権セットの遷移

ステップ	*temp* セット	許可された特権セット	実効特権セット	制限特権セット
初期状態	–	すべて	すべて	すべて
5行目 – *temp* を基本特権に設定します。	基本	すべて	すべて	すべて
7行目 – `file_dac_read` を *temp* に追加します。	基本 + `file_dac_read`	すべて	すべて	すべて
9行目 – `proc_exec` を *temp* から削除します。	基本 + `file_dac_read` – `proc_exec`	すべて	すべて	すべて
11行目 – *temp* を反転します。	すべて – (基本 + `file_dac_read` – `proc_exec`)	すべて	すべて	すべて
14行目 – 許可されたセット内の不要な特権をオフにします。	すべて – (基本 + `file_dac_read` – `proc_exec`)	基本 + `file_dac_read` – `proc_exec`	基本 + `file_dac_read` – `proc_exec`	すべて
16行目 – 制限セット内の不要な特権をオフにします。	すべて – (基本 + `file_dac_read` – `proc_exec`)	基本 + `file_dac_read` – `proc_exec`	基本 + `file_dac_read` – `proc_exec`	基本 + `file_dac_read` – `proc_exec`
18行目 – *temp* ファイルを解放します。	–	基本 + `file_dac_read` – `proc_exec`	基本 + `file_dac_read` – `proc_exec`	基本 + `file_dac_read` – `proc_exec`
22行目 – 必要になるまで `file_dac_read` をオフにします。	–	基本 – `proc_exec`	基本 – `proc_exec`	基本 + `file_dac_read` – `proc_exec`
24行目 – 必要に応じて `file_dac_read` をオンにします。	–	基本 + `file_dac_read` – `proc_exec`	基本 + `file_dac_read` – `proc_exec`	基本 + `file_dac_read` – `proc_exec`
27行目 – `read()` 操作後に`file_dac_read` をオフにします。	–	基本 – `proc_exec`	基本 – `proc_exec`	基本 + `file_dac_read` – `proc_exec`
29行目 – 不要になった `file_dac_read` をすべてのセットから削除します。	–	基本 – `proc_exec`	基本 – `proc_exec`	基本 – `proc_exec`

特権付きアプリケーション開発のガイドライン

ここでは、特権付きアプリケーションを開発する際の注意点を次に列挙します。

- 隔離されたシステムの使用: 特権付きアプリケーションのデバッグは本稼働システム上で決して行うべきではありません。不完全な特権付きアプリケーションを実行すると、セキュリティーが損なわれる可能性があります。
- 適切な ID 設定: 呼び出し元のプロセスの実効セット内に `proc_setid` 特権が含まれていないと、そのプロセスはユーザー ID、グループ ID、または追補グループ ID を変更できません。
- 特権の囲い込みの使用: アプリケーションが特権を使用する場合、システムのセキュリティーポリシーよりも優先されます。機密情報を危険にさらすことのないよう、囲い込みを使って特権操作を注意深く制御する必要があります。特権を囲い込みする方法については、32 ページの「特権のコーディング例」を参照してください。
- 基本特権から始める: 最小限の操作には基本特権が必要です。特権付きアプリケーションは基本セットから始める必要があります。その後、アプリケーションは必要に応じて特権の削除/追加を行う必要があります。一般的な起動シナリオを、次に示します。

  1. デーモンがスーパーユーザーとして起動します。
  2. デーモン内で基本特権セットがオンになります。
  3. デーモン内で、PRIV_FILE_LINK_ANY など、不要な基本特権のすべてがオフになります。
  4. デーモン内で、PRIV_FILE_DAC_READ など、その他の必要な特権が追加されます。
  5. UID がデーモンの UID に切り替わります。

- シェルエスケープの回避: シェルエスケープ内の新しいプロセスは、親プロセスの継承可能セットに含まれるすべての特権を使用できます。このため、一般ユーザーがシェルエスケープ経由で、セキュリティーを侵害する可能性があります。たとえば、一部のメールアプリケーションは、`!command` 行をコマンドと解釈し、それを実行します。したがって、一般ユーザーは、メールアプリケーションの任意の特権を利用するスクリプトを作成できます。不要なシェルエスケープを削除することをお勧めします。

承認について

承認は、`/etc/security/auth_attr` ファイル内に格納されます。承認を使用するアプリケーションを開発するには、次の手順に従います。

1. `/etc/security/auth_attr` 内で、1 つまたは複数の適切な承認を検索します。
2. プログラム開始時に、`chkauthattr(3SECDB)` 関数を使って必要な承認の有無を検査します。`chkauthattr()` 関数は、次の場所で順に承認を検索します。

- policy.conf(4) データベース内の AUTHS_GRANTED キー – AUTHS_GRANTED は、デフォルトで割り当てられた承認を示します。
- policy.conf(4) データベース内の PROFS_GRANTED キー – PROFS_GRANTED は、デフォルトで割り当てられた権利プロファイルを示します。chkauthattr() は、これらの権利プロファイルに指定された承認が含まれているかどうかを検査します。
- user_attr(4) データベース – このデータベースには、ユーザーに割り当てられたセキュリティー属性が格納されています。
- prof_attr(4) データベース – このデータベースには、ユーザーに割り当てられた権利プロファイルが格納されています。

chkauthattr() 実行時に正しい承認がこれらのどの場所にも見つからなかった場合、そのユーザーはプログラムへのアクセスを拒否されます。

3. このアプリケーションで必要な承認を管理者に知らせます。管理者への通知は、マニュアルページやその他の文書を通じて行えます。

例 2–3 承認の検査

次のコードは、chkauthattr() 関数を使ってあるユーザーの承認を検査する方法を示したものです。この場合、プログラムは solaris.job.admin 承認の有無を検査しています。このユーザーがこの承認を持っている場合、このユーザーはほかのユーザーのファイルを読み書きできます。この承認がない場合、このユーザーが操作できるのは、自身が所有するファイルだけになります。

```
/* Define override privileges */
priv_set_t *override_privs = priv_allocset();

/* Clear privilege set before adding privileges. */
priv_set(PRIV_OFF, PRIV_EFFECTIVE, PRIV_FILE_DAC_READ,
            priv_FILE_DAC_WRITE, NULL);

priv_addset(override_privs, PRIV_FILE_DAC_READ);
priv_addset(override_privs, PRIV_FILE_DAC_WRITE);

if (!chkauthattr("solaris.jobs.admin", username)) {
    /* turn off privileges */
    setppriv(PRIV_OFF, PRIV_EFFECTIVE, override_privs);
}
/* Authorized users continue to run with privileges */
/* Other users can read or write to their own files only */
```

◆ ◆ ◆ 第 3 章

PAM アプリケーションおよび PAM サービスの記述

プラグイン可能な認証モジュール (Pluggable Authentication Modules、PAM) は、システムエントリアプリケーションに対し、認証および関連セキュリティーサービスを提供します。この章の対象読者は、認証、アカウント管理、セッション管理、およびパスワード管理を PAM モジュール経由で行いたいと考えているシステムエントリアプリケーション開発者です。また、ここでは、PAM サービスモジュールの設計者向けの情報も記載します。この章では、次の内容について説明します。

PAM は最初、Sun によって開発されました。その後、X/Open (現在の Open Group) に PAM 仕様が提出されました。PAM 仕様は、『X/Open Single Sign-On Service (XSSO) - Pluggable Authentication』 (Open Group, UK ISBN 1-85912-144-6 June 1997) として入手可能となっています。PAM の Solaris 実装については、`pam(3PAM)`、`libpam(3LIB)`、および `pam_sm(3PAM)` のマニュアルページを参照してください。

PAM フレームワークの概要

PAM フレームワークは次の 4 つの部分から成ります。

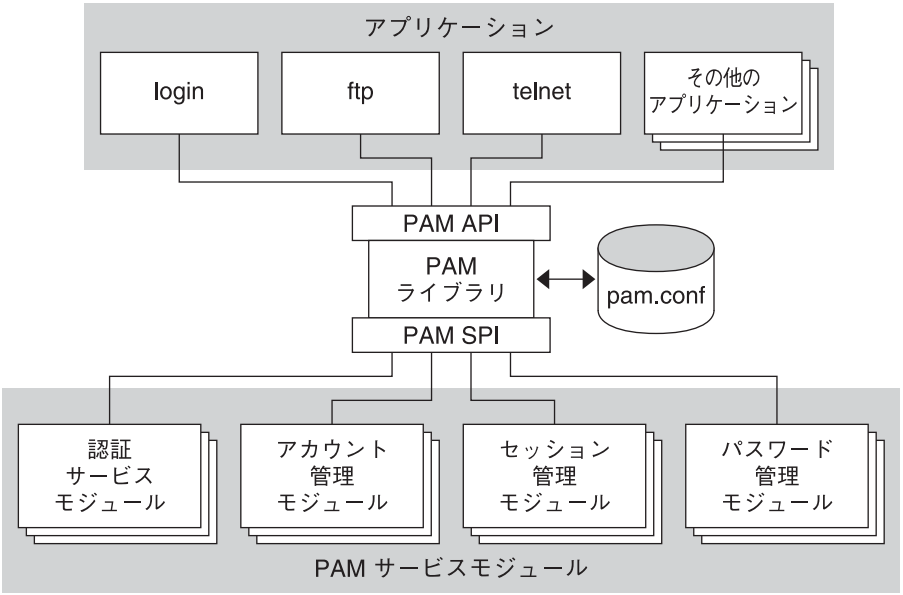
- PAM コンシューマ
- PAM ライブラリ
- `pam.conf(4)` 構成ファイル
- PAM サービスモジュール。プロバイダとも呼ばれる

このフレームワークは、認証関連アクティビティーの統一的な実施手段を提供します。このアプローチを使えば、アプリケーション開発者は、PAM サービスのポリシーの意味を知らなくてもサービスを使用できるようになります。アルゴリズムは一元的に提供されます。アルゴリズムの変更は、個々のアプリケーションとは無関

係に行えます。PAM を使えば、管理者は、アプリケーションを変更しないで、特定システムのニーズに合わせて認証プロセスを調整できるようになります。この調整は、PAM 構成ファイル `pam.conf` を通じて行われます。

次の図は、PAM のアーキテクチャーを示したものです。アプリケーションは、PAM アプリケーションプログラミングインタフェース (API) 経由で PAM ライブラリと通信します。PAM モジュールは、PAM サービスプロバイダインタフェース (SPI) 経由で PAM ライブラリと通信します。したがって、PAM ライブラリを使えば、アプリケーションとモジュールとの相互通信を実現できます。

図 3–1 PAM のアーキテクチャー



PAM サービスモジュール

PAM サービスモジュールは、`login`、`rlogin`、`telnet` などのシステムエントリアプリケーションに対し、認証およびその他のセキュリティーサービスを提供する共有ライブラリです。PAM サービスには次の 4 種類があります。

- 認証サービスモジュール – アカウントまたはサービスへのアクセス許可をユーザーに与えるためのモジュール。このサービスを提供するモジュールは、ユーザーの認証およびユーザー資格の設定を行います。

- アカウント管理モジュール – 現在のユーザーのアカウントが有効かどうかを決定するためのモジュール。このサービスを提供するモジュールは、パスワードまたはアカウントの有効期間を検査できるほか、時間制限付きアクセスの検査も行えます。
- セッション管理モジュール – ログインセッションの設定と終了を行うためのモジュール。
- パスワード管理モジュール – パスワードの強度規則の適用と認証トークンの更新を行うためのモジュール。

1 つの PAM モジュールには、上記サービスを 1 つ以上実装できます。ただし、作業内容が明確に定義された単純なモジュールを使用すれば、設定の柔軟性が増します。したがって、PAM サービスは個別のモジュールとして実装することをお勧めします。各サービスは必要に応じて使用できます。それには、pam.conf(4) ファイル内で必要な定義を行います。

たとえば、Solaris OS に付属する pam_authtok_check(5) モジュールを使えば、システム管理者はサイトのパスワードポリシーを設定できます。pam_authtok_check(5) モジュールは、提案されたパスワードを、さまざまな強度条件に基づいて検査します。

Solaris PAM モジュールの完全な一覧については、『man pages section 5: Standards, Environments, and Macros』を参照してください。PAM モジュールの接頭辞は pam_ です。

PAM ライブラリ

PAM ライブラリ libpam(3LIB) は、PAM アーキテクチャーの主要構成要素です。

- libpam は pam(3PAM) API をエクスポートします。アプリケーションは、この API を呼び出すことで、認証、アカウント管理、資格の確立、セッション管理、およびパスワード変更を行えます。
- libpam はマスター構成ファイル pam.conf(4) をインポートします。PAM 構成ファイルでは、利用可能なサービスごとに PAM モジュール要件が指定されます。pam.conf はシステム管理者によって管理されます。
- libpam は、サービスモジュールによってエクスポートされた pam_sm(3PAM) SPI をインポートします。

PAM 認証プロセス

コンシューマが PAM ライブラリを使ってユーザー認証を行う例として、login がユーザー認証を行う手順を次に示します。

1. login アプリケーションは、PAM セッションを開始するために、pam_start(3PAM) を呼び出し、login サービスを指定します。

2. アプリケーションは、PAM ライブラリ libpam(3LIB) によってエクスポートされた PAM API に含まれる pam_authenticate(3PAM) を呼び出します。
3. ライブラリは、pam.conf ファイル内で login エントリを検索します。
4. PAM ライブラリは、pam.conf 内で login サービス用として構成されたモジュールごとに、pam_sm_authenticate(3PAM) を呼び出します。pam_sm_authenticate() は PAM SPI に含まれる関数です。pam.conf 制御フラグと各呼び出しの結果によって、ユーザーがシステムへのアクセスを許可されるかどうかが決まります。このプロセスについての詳細は、『Solaris のシステム管理 (セキュリティサービス)』の「PAM の構成 (参照)」を参照してください。

PAM ライブラリはこのような方法で、PAM アプリケーションを、システム管理者によって構成された PAM モジュールへと接続します。

PAM コンシューマの要件

PAM コンシューマは PAM ライブラリ libpam とリンクする必要があります。各モジュールが提供するサービスをアプリケーションから利用するには、pam_start(3PAM) を呼び出して PAM ライブラリのインスタンスを初期化する必要があります。pam_start() を呼び出すと、ハンドルが初期化されます。このハンドルは、後続の PAM 呼び出し時に毎回指定する必要があります。アプリケーション内での PAM サービスの利用を終了する際には、pam_end() を呼び出し、PAM ライブラリが使用したすべてのデータをクリーンアップします。

PAM アプリケーションと PAM モジュール間の通信は、「アイテム」経由で実現されます。たとえば、初期化時に使うと便利なアイテムを、次に示します。

- PAM_USER – 現在の認証ユーザー
- PAM_AUTHTOK – パスワード
- PAM_USER_PROMPT – ユーザー名プロンプト
- PAM_TTY – ユーザーが通信に使用している端末
- PAM_REPOSITORY – ユーザーのアカウントリポジトリに対するすべての制限
- PAM_REPOSITORY – ユーザーのアカウントリポジトリに対するすべての制限
- PAM_RESOURCE – リソースに対するすべての制御

利用可能なすべてのアイテムの一覧については、pam_set_item(3PAM) のマニュアルページを参照してください。アプリケーションからアイテムを設定するには、pam_set_item (3PAM) を使用します。モジュールによって設定された値をアプリケーション内で取り出すには、pam_get_item(3PAM) を使用します。ただし、PAM_AUTHTOK と PAM_OLDAUTHTOK はアプリケーションから取得できません。また、PAM_SERVICE アイテムは設定できません。

PAM 構成

login、rlogin、su、cron などのシステムサービスに対する PAM サービスモジュールを構成するには、PAM 構成ファイル pam.conf(4) を使用します。システム管理者がこのファイルを管理します。pam.conf 内のエントリの順番が間違っていると、予期しない副作用が生じる可能性があります。たとえば、pam.conf の設定が不適切であると、ユーザーがロックアウトされ、修復のためにシングルユーザーモードが必要になる可能性があります。PAM 構成については、『Solaris のシステム管理 (セキュリティサービス)』の「PAM の構成 (参照)」を参照してください。

PAM サービスを使用するアプリケーションの記述

ここでは、いくつかの PAM 関数を使用するアプリケーション例を示します。

単純な PAM コンシューマ例

次の PAM コンシューマアプリケーションは、例示目的で提供されています。この例は、端末へのアクセスを試みるユーザーを検証する基本的な端末ロックアプリケーションです。この例では、次の手順を実行します。

1. PAM セッションを初期化します。

   PAM セッションは、pam_start(3PAM) 関数の呼び出しによって初期化されます。PAM コンシューマアプリケーションは、ほかの PAM 関数を呼び出す前に PAM セッションを最初に確立する必要があります。pam_start(3PAM) 関数は、次の引数を使用します。

   - plock – サービス名、つまり、アプリケーションの名前。サービス名は、PAM フレームワークで構成ファイル /etc/pam.conf 内のどの規則が適切かを決定するために使用されます。通常、サービス名はロギングとエラーレポートに使用されます。
   - pw->pw_name – ユーザー名は、PAM フレームワークの対象となるユーザーの名前です。
   - &conv – 対話関数 conv は、PAM がユーザーまたはアプリケーションと通信するための汎用手段を提供します。PAM モジュールは通信の実施方法を認識する方法を持っていないため、対話関数が必要です。通信は、GUI、コマンド行、スマートカードリーダー、またはその他のデバイスを使用して行うことができます。詳細は、47 ページの「対話関数の記述」を参照してください。
   - &pamh – PAM ハンドル pamh は、PAM フレームワークで現在の処理に関する情報を格納する際に使用される不透明なハンドルです。このハンドルは、pam_start() の呼び出しが成功することによって返されます。

注 – PAM インタフェースを呼び出すアプリケーションは、認証、パスワードの変更、プロセス資格操作、監査状態の初期化など、すべての必要な処理を実行するための十分な特権を持っている必要があります。この例では、ローカルユーザーのパスワードを検証するために、アプリケーションは /etc/shadow を読み取り可能でなければなりません。

2. ユーザーを認証します。

   アプリケーションは、pam_authenticate(3PAM) を呼び出して現在のユーザーを認証します。一般に、ユーザーは認証サービスの種類に応じて、パスワードまたはその他の認証トークンを入力する必要があります。PAM フレームワークは /etc/pam.conf 内の認証サービス auth にリストされているモジュールを起動します。サービス名 plock は、使用する pam.conf エントリを決定する際に使用されます。plock のエントリが存在しない場合は、デフォルトで other のエントリが使用されます。アプリケーション構成ファイルで NULL パスワードが明示的に禁じられている場合は、PAM_DISALLOW_NULL_AUTHTOK フラグが渡されます。Solaris アプリケーションは、/etc/default/login の PASSREQ=YES 設定を確認します。

3. アカウントの有効性を確認します。

   例では、pam_acct_mgmt(3PAM) 関数を使用して、認証ユーザーのアカウントの有効性を確認します。この例では、pam_acct_mgmt() がパスワードの有効期限を確認します。

   pam_acct_mgmt() 関数は、PAM_DISALLOW_NULL_AUTHTOK フラグも使用します。pam_acct_mgmt() が PAM_NEW_AUTHTOK_REQD を返す場合は、認証ユーザーにパスワードの変更を許可するために、pam_chauthtok(3PAM) が呼び出されます。

4. パスワードの有効期限が切れていることがシステムによって検出された場合は、ユーザーにパスワードの変更を強制します。

   例では、「成功」が返されるまでループを使用して pam_chauthtok() を呼び出します。ユーザーが認証情報 (通常はパスワード) の変更に成功した場合、pam_chauthtok() 関数は「成功」を返します。この例では、「成功」が返されるまでループが継続します。通常はアプリケーションで、終了するまでの最大試行回数を設定します。

5. pam_setcred(3PAM) を呼び出します。

   pam_setcred(3PAM) 関数は、ユーザー資格の確立、変更、削除を行う際に使用されます。pam_setcred() は通常、ユーザー認証の完了時に呼び出されます。この呼び出しは、アカウントの検証完了後、セッションのオープン前に行われます。新しいユーザーセッションを確立する場合は、pam_setcred() 関数で PAM_ESTABLISH_CRED フラグを使用します。lockscreen の場合のように、セッションが既存セッションの更新版である場合、pam_setcred() を PAM_REFRESH_CRED フラグとともに呼び出すべきです。su を使用したり特定の

役割を引き受けたりする場合のように、セッションが資格を変更する場合、pam_setcred() を PAM_REINITIALIZE_CRED フラグとともに呼び出すべきです。

6. PAM セッションをクローズします。

   PAM セッションは、pam_end(3PAM) 関数の呼び出しによってクローズします。また、pam_end() は、すべての PAM リソースを解放します。

この PAM コンシューマアプリケーション例のソースコードを、次に示します。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

例 3–1 PAM コンシューマアプリケーションの例

```
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <strings.h>
#include <signal.h>
#include <pwd.h>
#include <errno.h>
#include <security/pam_appl.h>

extern int pam_tty_conv(int num_msg, struct pam_message **msg,
         struct pam_response **response, void *appdata_ptr);

/* Disable keyboard interrupts (Ctrl-C, Ctrl-Z, Ctrl-\) */
static void
disable_kbd_signals(void)
{
    (void) signal(SIGINT, SIG_IGN);
    (void) signal(SIGTSTP, SIG_IGN);
    (void) signal(SIGQUIT, SIG_IGN);
}

/* Terminate current user session, i.e., logout */
static void
logout()
{
    pid_t pgroup = getpgrp();

    (void) signal(SIGTERM, SIG_IGN);
    (void) fprintf(stderr, "Sorry, your session can't be restored.\n");
    (void) fprintf(stderr, "Press return to terminate this session.\n");
    (void) getchar();
    (void) kill(-pgroup, SIGTERM);
```

例 3–1 PAM コンシューマアプリケーションの例 *(続き)*

```
    (void) sleep(2);
    (void) kill(-pgroup, SIGKILL);
    exit(-1);
}

int
/*ARGSUSED*/
main(int argc, char *argv)
{
    struct pam_conv conv = { pam_tty_conv, NULL };
    pam_handle_t *pamh;
    struct passwd *pw;
    int err;

    disable_kbd_signals();
    if ((pw = getpwuid(getuid())) == NULL) {
        (void) fprintf(stderr, "plock: Can't get username: %s\n",
            strerror(errno));
        exit(1);
    }

    /* Initialize PAM framework */
    err = pam_start("plock", pw->pw_name, &conv, &pamh);
    if (err != PAM_SUCCESS) {
        (void) fprintf(stderr, "plock: pam_start failed: %s\n",
            pam_strerror(pamh, err));
        exit(1);
    }

    /* Authenticate user in order to unlock screen */
    do {
        (void) fprintf(stderr, "Terminal locked for %s. ", pw->pw_name);
        err = pam_authenticate(pamh, 0);
        if (err == PAM_USER_UNKNOWN) {
            logout();
        } else if (err != PAM_SUCCESS) {
            (void) fprintf(stderr, "Invalid password.\n");
        }
    } while (err != PAM_SUCCESS);

    /* Make sure account and password are still valid */
    switch (err = pam_acct_mgmt(pamh, 0)) {
    case PAM_SUCCESS:
        break;
    case PAM_USER_UNKNOWN:
    case PAM_ACCT_EXPIRED:
        /* User not allowed in anymore */
        logout();
        break;
    case PAM_NEW_AUTHTOK_REQD:
        /* The user's password has expired. Get a new one */
        do {
            err = pam_chauthtok(pamh, 0);
        } while (err == PAM_AUTHTOK_ERR);
        if (err != PAM_SUCCESS)
            logout();
```

例 3–1　PAM コンシューマアプリケーションの例　　(続き)

```
        break;
    default:
        logout();
    }

if (pam_setcred(pamh, PAM_REFRESH_CRED) != PAM_SUCCESS){
    logout();
}

    (void) pam_end(pamh, 0);
    return(0);
    /*NOTREACHED*/
}
```

その他の有用な PAM 関数

前記の単純なアプリケーション 例 3–1 では、主要 PAM 関数のうちほんの数種類しか使用されていません。ここでは、その他の有用な PAM 関数をいくつか紹介します。

pam_open_session(3PAM) 関数は、ユーザー認証が成功したあと、新しいセッションをオープンする際に呼び出されます。

pam_getenvlist(3PAM) 関数は、新しい環境を確立する際に呼び出されます。pam_getenvlist() は、既存環境にマージすべき新しい環境を返します。

対話関数の記述

PAM モジュール (アプリケーション) はいくつかの方法でユーザーと通信できます。たとえば、コマンド行を使用する方法や、ダイアログボックスを使用する方法などです。その結果、ユーザーと通信する PAM コンシューマの設計者は、「対話関数」と呼ばれるものを記述する必要があります。対話関数は、特定の通信手段に依存することなしに、ユーザーとモジュール間でメッセージの受け渡しを行います。対話関数は、対話関数コールバック関数の pam_message パラメータ内の *msg_style* パラメータから、メッセージタイプを得ます。pam_start(3PAM) のマニュアルページを参照してください。

開発者は、PAM とユーザー間の通信手段について、何らかの仮定を行なってはいけません。むしろ、アプリケーションは、処理が完了するまでユーザーとメッセージを交換し続ける必要があります。アプリケーションは、対話関数のメッセージ文字列を、解釈または変更することなしに表示する必要があります。個々のメッセージには、複数の行を含めることができるほか、制御文字や余分な空白も含めることができます。対話関数に送信する文字列を各言語対応にすることは、サービスモジュールの責任であることに注意してください。

対話関数の例 pam_tty_conv() を、次に示します。pam_tty_conv() の引数は次のとおりです。

- *num_msg* – この関数に渡されるメッセージの数。
- ***mess* – ユーザーからのメッセージを格納するバッファへのポインタ
- ***resp* – ユーザーへの応答を格納するバッファへのポインタ。
- **my_data* – アプリケーションデータへのポインタ。

この関数例は、`stdin` からユーザー入力を取得します。応答バッファーに対するメモリーの割り当ては、このルーチンが行う必要があります。最大値 PAM_MAX_NUM_MSG を設定すれば、メッセージの数を制限できます。対話関数がエラーを返す場合、対話関数は応答に割り当てられていたすべてのメモリーを消去および解放する役割を担います。さらに、対話関数は応答ポインタを NULL に設定する必要があります。メモリーの消去は、ゼロ埋めアプローチを使用して完了されるべきです。対話関数の呼び出し側には、呼び出し側に返されたすべての応答を解放する責任があります。対話を実現するために、この関数は、ユーザーアプリケーションからのメッセージをループ処理します。有効なメッセージは `stdout` に書き込まれ、エラーメッセージは `stderr` に書き込まれます。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。`http://www.sun.com/download/products.xml?id=41912db5` を参照してください。

例 3–2　PAM 対話関数

```
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma ident    "@(#)pam_tty_conv.c    1.4    05/02/12 SMI"

#define    __EXTENSIONS__    /* to expose flockfile and friends in stdio.h */
#include <errno.h>
#include <libgen.h>
#include <malloc.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <stropts.h>
#include <unistd.h>
#include <termio.h>
#include <security/pam_appl.h>

static int ctl_c;    /* was the conversation interrupted? */

/* ARGSUSED 1 */
static void
interrupt(int x)
{
    ctl_c = 1;
}

/* getinput -- read user input from stdin abort on ^C
```

例 3–2 PAM 対話関数 *(続き)*

```
 *    Entry    noecho == TRUE, don't echo input.
 *    Exit    User's input.
 *        If interrupted, send SIGINT to caller for processing.
 */
static char *
getinput(int noecho)
{
    struct termio tty;
    unsigned short tty_flags;
    char input[PAM_MAX_RESP_SIZE];
    int c;
    int i = 0;
    void (*sig)(int);

    ctl_c = 0;
    sig = signal(SIGINT, interrupt);
    if (noecho) {
        (void) ioctl(fileno(stdin), TCGETA, &tty);
        tty_flags = tty.c_lflag;
        tty.c_lflag &= ~(ECHO | ECHOE | ECHOK | ECHONL);
        (void) ioctl(fileno(stdin), TCSETAF, &tty);
    }

    /* go to end, but don't overflow PAM_MAX_RESP_SIZE */
    flockfile(stdin);
    while (ctl_c == 0 &&
        (c = getchar_unlocked()) != '\n' &&
        c != '\r' &&
        c != EOF) {
        if (i < PAM_MAX_RESP_SIZE) {
            input[i++] = (char)c;
        }
    }
    funlockfile(stdin);
    input[i] = '\0';
    if (noecho) {
        tty.c_lflag = tty_flags;
        (void) ioctl(fileno(stdin), TCSETAW, &tty);
        (void) fputc('\n', stdout);
    }
    (void) signal(SIGINT, sig);
    if (ctl_c == 1)
        (void) kill(getpid(), SIGINT);

    return (strdup(input));
}

/* Service modules do not clean up responses if an error is returned.
 * Free responses here.
 */
static void
free_resp(int num_msg, struct pam_response *pr)
{
    int i;
    struct pam_response *r = pr;
```

例 3–2 PAM対話関数 (続き)

```
    if (pr == NULL)
        return;

    for (i = 0; i < num_msg; i++, r++) {

        if (r->resp) {
            /* clear before freeing -- may be a password */
            bzero(r->resp, strlen(r->resp));
            free(r->resp);
            r->resp = NULL;
        }
    }
    free(pr);
}

/* ARGSUSED */
int
pam_tty_conv(int num_msg, struct pam_message **mess,
    struct pam_response **resp, void *my_data)
{
    struct pam_message *m = *mess;
    struct pam_response *r;
    int i;

    if (num_msg <= 0 || num_msg >= PAM_MAX_NUM_MSG) {
        (void) fprintf(stderr, "bad number of messages %d "
            "<= 0 || >= %d\n",
            num_msg, PAM_MAX_NUM_MSG);
        *resp = NULL;
        return (PAM_CONV_ERR);
    }
    if ((*resp = r = calloc(num_msg,
        sizeof (struct pam_response))) == NULL)
        return (PAM_BUF_ERR);

    /* Loop through messages */
    for (i = 0; i < num_msg; i++) {
        int echo_off;

        /* bad message from service module */
        if (m->msg == NULL) {
            (void) fprintf(stderr, "message[%d]: %d/NULL\n",
                i, m->msg_style);
            goto err;
        }

        /*
         * fix up final newline:
         *     removed for prompts
         *     added back for messages
         */
        if (m->msg[strlen(m->msg)] == '\n')
            m->msg[strlen(m->msg)] = '\0';

        r->resp = NULL;
        r->resp_retcode = 0;
```

例 3–2 PAM 対話関数 *(続き)*

```
        echo_off = 0;
        switch (m->msg_style) {

        case PAM_PROMPT_ECHO_OFF:
            echo_off = 1;
            /*FALLTHROUGH*/

        case PAM_PROMPT_ECHO_ON:
            (void) fputs(m->msg, stdout);

            r->resp = getinput(echo_off);
            break;

        case PAM_ERROR_MSG:
            (void) fputs(m->msg, stderr);
            (void) fputc('\n', stderr);
            break;

        case PAM_TEXT_INFO:
            (void) fputs(m->msg, stdout);
            (void) fputc('\n', stdout);
            break;

        default:
            (void) fprintf(stderr, "message[%d]: unknown type "
                "%d/val=\"%s\"\n",
                i, m->msg_style, m->msg);
            /* error, service module won't clean up */
            goto err;
        }
        if (errno == EINTR)
            goto err;

        /* next message/response */
        m++;
        r++;
    }
    return (PAM_SUCCESS);

err:
    free_resp(i, r);
    *resp = NULL;
    return (PAM_CONV_ERR);
}
```

PAM サービスを提供するモジュールの記述

ここでは、PAM サービスモジュールの記述方法を説明します。

PAM サービスプロバイダの要件

PAM サービスモジュールは、pam_get_item(3PAM) と pam_set_item(3PAM) を使ってアプリケーションとの通信を行います。サービスモジュール同士の通信には、pam_get_data(3PAM) と pam_set_data(3PAM) が使用されます。同一プロジェクト内のサービスモジュール間でデータを交換する必要がある場合、そのプロジェクト内で一意に決まるデータ名を確立する必要があります。その後、サービスモジュールは、関数 pam_get_data() と pam_set_data() を使ってそのデータを共有できます。

サービスモジュールは、次の 3 種類の PAM 戻りコードのいずれかを返す必要があります。

- PAM_SUCCESS: 要求されているポリシーに合致しているという肯定的な決定を、モジュールが行なった場合。
- PAM_IGNORE: モジュールがポリシー決定を行わなかった場合。
- PAM_*error*: モジュールが参加していた決定が失敗した場合。*error* は、汎用エラーコード、サービスモジュールタイプに固有のコードのいずれかです。別のサービスモジュールタイプのエラーコードは使えません。エラーコードについては、pam_sm_*module-type* のマニュアルページを参照してください。

1 つのサービスモジュール内に複数の機能が含まれている場合、それらの機能はそれぞれ個別のモジュールに分割することをお勧めします。そうすることで、システム管理者は、ポリシー設定時によりきめ細かい制御を行えるようになります。

新しいサービスモジュールを作成した場合、対応するマニュアルページを提供する必要があります。マニュアルページには、次の情報を含める必要があります。

- モジュールが受け入れる引数。
- モジュールが実装しているすべての関数。
- アルゴリズムに対するフラグの効果。
- 必要とされるすべての PAM アイテム。
- このモジュールに固有のエラー戻りコード。

サービスモジュールは、メッセージを抑制するための PAM_SILENT フラグを尊重することが求められます。デバッグ情報を syslog に記録するには、*debug* 引数を指定することをお勧めします。デバッグ情報を記録するには、syslog(3C) 使用時に LOG_AUTH と LOG_DEBUG を指定します。その他のメッセージは、LOG_AUTH と適切な優先度を指定して syslog() に送るべきです。openlog(3C)、closelog(3C)、および setlogmask(3C) という 3 つの関数はアプリケーションの設定に悪影響を与えるので、決して使用しないでください。

PAM プロバイダサービスモジュールの例

ここでは、PAM サービスモジュールの例を示します。この例では、ユーザーがこのサービスへのアクセスが許可されているグループのメンバーかどうかを確認します。プロバイダは、成功の場合にアクセスを許可し、失敗の場合にエラーメッセージをログに記録します。この例では、次の手順を実行します。

1. `/etc/pam.conf` の構成行からこのモジュールに渡されたオプションを解析します。

   このモジュールは、`nowarn` オプションおよび `debug` オプション、さらに固有のオプション `group` を受け入れます。`group` オプションを使用する場合、デフォルトで使用されるグループ `root` 以外の特定のグループに対してアクセスを許可するようにモジュールを構成できます。この例については、ソースコードの DEFAULT_GROUP の定義を参照してください。たとえば、グループ `staff` に属するユーザーによる `telnet(1)` アクセスを許可するには、`/etc/pam.conf` の `telnet` スタックにある次の行を使用できます。

   ```
   telnet  account  required  pam_members_only.so.1 group=staff
   ```

2. ユーザー名、サービス名、およびホスト名を取得します。

   ユーザー名は、現在のユーザー名を PAM ハンドルから取り出す、`pam_get_user(3PAM)` の呼び出しによって取得されます。ユーザー名が設定されていない場合、アクセスは拒否されます。サービス名とホスト名は、`pam_get_item(3PAM)` の呼び出しによって取得されます。

3. 有効にする情報を検証します。

   ユーザー名が設定されていない場合、アクセスは拒否されます。有効にするグループが定義されていない場合、アクセスは拒否されます。

4. 現在のユーザーが、このホストへのアクセスが許可されている特殊グループのメンバーであることを確認し、アクセスを許可します。

   固有のグループが定義されていてもメンバーが 1 つも含まれていない場合は、このモジュールがどのアカウント検証プロセスにも参加しないことを示す、PAM_IGNORE が返されます。決定は、スタック上のほかのモジュールに委ねられます。

5. ユーザーが特殊グループのメンバーではない場合は、アクセスが拒否されたことをユーザーに知らせるメッセージを表示します。

   メッセージをログに記録してこのイベントを記録します。

次の例は、PAM プロバイダ例のソースコードです。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。`http://www.sun.com/download/products.xml?id=41912db5` を参照してください。

例 3–3 PAM サービスモジュール例

```
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <stdio.h>
#include <stdlib.h>
#include <grp.h>
#include <string.h>
#include <syslog.h>
#include <libintl.h>
#include <security/pam_appl.h>

/*
 * by default, only users who are a member of group "root" are allowed access
 */
#define    DEFAULT_GROUP "root"

static char *NOMSG =
    "Sorry, you are not on the access list for this host - access denied.";

int
pam_sm_acct_mgmt(pam_handle_t * pamh, int flags, int argc, const char **argv)
{
    char *user = NULL;
    char *host = NULL;
    char *service = NULL;
    const char *allowed_grp = DEFAULT_GROUP;
    char grp_buf[4096];
    struct group grp;
    struct pam_conv *conversation;
    struct pam_message message;
    struct pam_message *pmessage = &message;
    struct pam_response *res = NULL;
    int i;
    int nowarn = 0;
    int debug = 0;

    /* Set flags to display warnings if in debug mode. */
    for (i = 0; i < argc; i++) {
        if (strcasecmp(argv[i], "nowarn") == 0)
            nowarn = 1;
        else if (strcasecmp(argv[i], "debug") == 0)
            debug = 1;
        else if (strncmp(argv[i], "group=", 6) == 0)
            allowed_grp = &argv[i][6];
    }
    if (flags & PAM_SILENT)
        nowarn = 1;

    /* Get user name,service name, and host name. */
    (void) pam_get_user(pamh, &user, NULL);
    (void) pam_get_item(pamh, PAM_SERVICE, (void **) &service);
    (void) pam_get_item(pamh, PAM_RHOST, (void **) &host);

    /* Deny access if user is NULL. */
    if (user == NULL) {
```

例 3–3 PAM サービスモジュール例 (続き)

```
        syslog(LOG_AUTH|LOG_DEBUG,
            "%s: members_only: user not set", service);
        return (PAM_USER_UNKNOWN);
    }

    if (host == NULL)
        host = "unknown";

    /*
     * Deny access if vuser group is required and user is not in vuser
     * group
     */
    if (getgrnam_r(allowed_grp, &grp, grp_buf, sizeof (grp_buf)) == NULL) {
        syslog(LOG_NOTICE|LOG_AUTH,
            "%s: members_only: group \"%s\" not defined",
            service, allowed_grp);
        return (PAM_SYSTEM_ERR);
    }

    /* Ignore this module if group contains no members. */
    if (grp.gr_mem[0] == 0) {
        if (debug)
            syslog(LOG_AUTH|LOG_DEBUG,
                "%s: members_only: group %s empty: "
                "all users allowed.", service, grp.gr_name);
        return (PAM_IGNORE);
    }

    /* Check to see if user is in group. If so, return SUCCESS. */
    for (; grp.gr_mem[0]; grp.gr_mem++) {
        if (strcmp(grp.gr_mem[0], user) == 0) {
            if (debug)
                syslog(LOG_AUTH|LOG_DEBUG,
                    "%s: user %s is member of group %s. "
                    "Access allowed.",
                    service, user, grp.gr_name);
            return (PAM_SUCCESS);
        }
    }

    /*
     * User is not a member of the group.
     * Set message style to error and specify denial message.
     */
    message.msg_style = PAM_ERROR_MSG;
    message.msg = gettext(NOMSG);

    /* Use conversation function to display denial message to user. */
    (void) pam_get_item(pamh, PAM_CONV, (void **) &conversation);
    if (nowarn == 0 && conversation != NULL) {
        int err;
        err = conversation->conv(1, &pmessage, &res,
            conversation->appdata_ptr);
        if (debug && err != PAM_SUCCESS)
            syslog(LOG_AUTH|LOG_DEBUG,
                "%s: members_only: conversation returned "
```

例 3–3 PAM サービスモジュール例 *(続き)*

```
                "error %d (%s).", service, err,
                pam_strerror(pamh, err));

        /* free response (if any) */
        if (res != NULL) {
            if (res->resp)
                free(res->resp);
            free(res);
        }
    }

    /* Report denial to system log and return error to caller. */
    syslog(LOG_NOTICE | LOG_AUTH, "%s: members_only: "
        "Connection for %s not allowed from %s", service, user, host);

    return (PAM_PERM_DENIED);
}
```

◆ ◆ ◆ 第 4 章

GSS-API を使用するアプリケーションの記述

Generic Security Service Application Programming Interface (GSS-API) は、ピアとなるアプリケーションに送信されるデータを保護する方法をアプリケーションに提供します。接続は通常、あるマシン上のクライアントから別のマシン上のサーバーに対して行われます。この章では、次の内容について説明します。

GSS-API の紹介

GSS-API を使用すると、プログラマはセキュリティーの点で汎用的なアプリケーションを記述できます。開発者は、特定のプラットフォーム、セキュリティー機構、保護の種類、または転送プロトコル向けにセキュリティー実装をカスタマイズする必要はありません。GSS-API を使用すれば、プログラマはネットワークデータを保護する方法の詳細を知る必要がありません。GSS-API を使用するプログラムは、ネットワークセキュリティーに関する移植性が高くなります。この移植性が、Generic Security Service API の優れた特徴を示します。

GSS-API は、セキュリティーサービスを汎用的な方法で呼び出し元に提供するフレームワークです。次の図に示すように、GSS-API フレームワークは、Kerberos v5 や公開鍵技術など、基盤となるさまざまな機構や技術によって支えられています。

図 4–1　GSS-API の層

GSS-API の主な機能は、簡単に言うと次の 2 つです。

1. GSS–API は、セキュリティー「コンテキスト」を作成し、アプリケーション間でのデータの送受信は、このコンテキスト内で行うことができます。コンテキストは、2 つのアプリケーションが互いに信頼している状態を表します。コンテキストを共有するアプリケーションは、相手がだれであるかを知っており、したがって、そのコンテキストが継続する限り、互いにデータを転送できます。
2. GSS–API は、「セキュリティーサービス」として知られる 1 種類以上の保護機能を、転送データに対して適用します。セキュリティーサービスについては、59 ページの「GSS-API のセキュリティーサービス」を参照してください。

さらに、GSS-API は次の機能を実行します。

- データ変換
- エラーの検査
- ユーザー特権の委託
- 情報の表示
- 識別情報の比較

GSS-API にはさまざまな補助関数や簡易関数が含まれています。

アプリケーションの移植性とGSS-API

GSS-APIは、アプリケーションに対して次のような移植性を提供します。

- 機構非依存。GSS-APIは汎用的なセキュリティーインタフェースを提供します。デフォルトのセキュリティー機構を指定することで、アプリケーションは適用すべき機構や機構の詳細を知る必要がなくなります。
- プロトコル非依存。GSS-APIは特定の通信プロトコルまたはプロトコル群に依存しません。たとえば、GSS–APIは、ソケット、RCP、TCP/IPのいずれを使用するアプリケーションからも使用可能です。

  RPCSEC_GSSは、GSS-APIとRPCをスムースに統合するために追加される層です。詳細は、60ページの「リモートプロシージャー呼び出しとGSS-API」を参照してください。
- プラットフォーム非依存。GSS-APIは、アプリケーションが動作しているオペレーティングシステムの種類に依存しません。
- 保護品質非依存。保護品質(Quality of Protection、QOP)とは、データを暗号化したり暗号タグを生成したりする際に使用されるアルゴリズムの種類を示します。GSS-APIでは、プログラマはQOPを無視できます。それには、GSS-APIが提供するデフォルトを使用します。一方、必要であればアプリケーションはQOPを指定することもできます。

GSS-APIのセキュリティーサービス

GSS-APIは次の3種類のセキュリティーサービスを提供します。

- 認証 – 認証は、GSS-APIによって提供される基本的なセキュリティー機能です。認証とは相手の身元を確認することです。あるユーザーが認証されると、システムは、そのユーザーがそのユーザー名で活動する権利を持つ人であるとみなします。
- 整合性 – 「整合性」は、データの有効性を検証することです。データが有効なユーザーから送られてきたとしても、そのデータ自体が破壊または改ざんされている可能性があります。整合性は、メッセージが完全に意図されたとおりの内容であり、情報の追加や削除がまったく行われていないことを保証します。GSS-APIでは、メッセージ整合性コード(Message Integrity Code、MIC)と呼ばれる暗号タグを、データに添付できるようになっています。MICは、ユーザーが受信したデータが、送信側が送信したデータと同一であることを証明します。
- 機密性 – 「機密性」は、メッセージを傍受した第三者がその内容を読み取ろうとしても読み取れないことを保証します。認証と整合性のどちらも、データに変更を施すわけではありません。したがって、データが何らかの方法で傍受された場合、そのデータの内容が他人に読み取られてしまいます。したがって、GSS-APIではデータを暗号化できるようになっています。ただしそれには、暗号化をサポートする機構が利用可能である必要があります。このようにデータを暗号化することを機密性と呼びます。

GSS-API で利用可能な機構

GSS-API の現在の実装では、次の機構が利用できます。 Kerberos v5、Diffie-Hellman、および SPNEGO。Kerberos 実装についての詳細は、『Solaris のシステム管理 (セキュリティサービス)』の第 21 章「Kerberos サービスについて」を参照してください。Kerberos v5 は、GSS-API 対応プログラムが動作するすべてのシステム上で、インストールおよび実行される必要があります。

リモートプロシージャー呼び出しと GSS-API

RPC (Remote Procedure Call) プロトコルをネットワークアプリケーションに使用するプログラマは、RPCSEC_GSS を使用してセキュリティーを提供できます。RPCSEC_GSS は GSS-API 上にある別の層です。RPCSEC_GSS は GSS-API のすべての機能を RPC 用にカスタマイズした形式で提供します。実際、RPCSC_GSS は GSS-API の多くの側面をプログラマから隠蔽する役割を果たしており、その結果、特に高い操作性と移植性を備えた RPC セキュリティーが実現されています。RPCSEC_GSS についての詳細は、『ONC+ 開発ガイド』の「RPCSEC_GSS を使用した認証」を参照してください。

次の図は、RPCSEC_GSS 層がアプリケーションと GSS-API の間に位置している様子を示したものです。

図 4–2 RPCSEC_GSS と GSS-API

GSS-APIの制限

GSS-API は、データ保護作業を単純化しますが、GSS-API の一般的な性質に合致しないいくつかの作業をサポートしていません。GSS-API が実行「しない」作業は、次のとおりです。

- ユーザーまたはアプリケーションにセキュリティー資格を提供すること。資格は、実際のセキュリティー機構が提供する必要があります。GSS-API は、アプリケーションが資格を自動的または明示的に獲得することを可能にしています。
- アプリケーション間でデータを転送すること。セキュリティー関連のデータまたは通常のデータのどちらの場合でも、ピア間ですべてのデータの転送を処理することはアプリケーションの責任です。
- 転送データのさまざまな種類を識別すること。たとえば、GSS-API は、データパケットがプレーンデータ、暗号化データのいずれであるかを認識できません。
- 非同期エラーによる状態を示すこと。
- マルチプロセスプログラムのプロセス間で送信される情報をデフォルトで保護すること。
- GSS-API 関数に渡される文字列バッファーを割り当てること。62 ページの「GSS-API の文字列とそれに類するデータ」を参照してください。
- GSS-API データ領域を解放すること。そうしたメモリー領域の解放は、`gss_release_buffer()` や `gss_delete_name()` などの関数を使って明示的に行う必要があります。

GSS-API の言語バインディング

このマニュアルでは現在、GSS-API の C 言語バインディング、つまり関数とデータ型だけに言及しています。Java バインディング版の GSS-API が利用可能になりました。Java GSS-API には、RFC 2853 で規定された Generic Security Services Application Program Interface (GSS-API) に対する Java バインディングが含まれています。

GSS-API に関する詳細情報の入手先

GSS-API についての詳細は、次の 2 つの文書を参照してください。

- 『Generic Security Service Application Program Interface』 (ftp://ftp.isi.edu/in-notes/rfc2743.txt) は、GSS-API の概念について簡単に説明しています。
- 『Generic Security Service API Version 2: C-Bindings』 (ftp://ftp.isi.edu/in-notes/rfc2744.txt) は、C 言語ベースの GSS-API に固有の情報を提供しています。

GSS-API の重要な要素

ここでは、GSS-API の重要な概念である、主体、GSS-API データ型、GSS-API 状態コード、および GSS-API トークンについて説明します。

GSS-API データ型

ここでは、主な GSS-API データ型について説明します。すべての GSS-API データ型を確認するには、234 ページの「GSS-API データ型と値」を参照してください。

GSS-API の整数

`int` のサイズはプラットフォームによって異なるため、GSS-API は次の整数型を提供します。`OM_uint32`。これは、32 ビットの符号なし整数です。

GSS-API の文字列とそれに類するデータ

GSS-API はすべてのデータを内部形式で処理するため、文字列も、GSS-API 形式に変換したあとで GSS-API 関数に渡す必要があります。GSS-API は、`gss_buffer_desc` 構造体を使って文字列を処理します。

```
typedef struct gss_buffer_desc_struct {
     size_t     length;
     void       *value;
}  gss_buffer_desc *gss_buffer_t;
```

gss_buffer_tは、そうした構造体へのポインタです。文字列は、GSS-API関数に渡す前にgss_buffer_desc構造体に変換しておく必要があります。次の例では、汎用的なGSS-API関数を使って、送信前のメッセージに保護を適用しています。

例4–1 GSS-APIにおける文字列の使用法

```
char *message_string;
gss_buffer_desc input_msg_buffer;

input_msg_buffer.value = message_string;
input_msg_buffer.length = strlen(input_msg_buffer.value) + 1;

gss_generic_function(arg1, &input_msg_buffer, arg2...);

gss_release_buffer(input_msg_buffer);
```

ここで、input_msg_bufferは、終了時にgss_release_buffer()を使って解放する必要がある点に注意してください。

gss_buffer_descオブジェクトは文字列だけに使用されるわけではありません。たとえば、トークンもgss_buffer_descオブジェクトとして処理されます。詳細は、72ページの「GSS-API トークン」を参照してください。

GSS-APIにおける名前

「名前」は主体を指します。ネットワークセキュリティーの用語では、「主体」とは、ユーザー、プログラム、またはマシンを指します。主体はクライアントまたはサーバーのどちらにでもなり得ます。主体の例を、次にいくつか挙げます。

- 別のマシンにログインするユーザー (*user@machine* など)
- ネットワークサービス (*nfs@machine* など)
- アプリケーションを実行するマシン (*myHost@eng.company.com* など)

GSS-APIでは、名前はgss_name_tオブジェクトとして格納されます。このオブジェクトはアプリケーションに対して不透明です。名前をgss_buffer_tオブジェクトからgss_name_t形式に変換するには、gss_import_name()関数を使用します。インポートされたすべての名前には関連する「名前型」が割り当てられます。名前型とは、その名前の形式を示すものです。名前型については、70ページの「GSS-API のOID」を参照してください。有効な名前型の一覧については、235ページの「名前型」を参照してください。

gss_import_name()の構文は次のとおりです。

```
OM_uint32 gss_import_name (
       OM_uint32          *minor-status,
```

```
const gss_buffer_t input-name-buffer,
const gss_OID      input-name-type,
gss_name_t         *output-name)
```

minor-status 実際の機構から戻される状態コード。71 ページの「GSS-API 状態コード」を参照してください。

input-name-buffer インポートされた名前が格納される gss_buffer_desc 構造体。この構造体は、アプリケーション側で明示的に割り当てる必要があります。62 ページの「GSS-API の文字列とそれに類するデータ」および例 4–2 を参照してください。この引数は、アプリケーションでの使用終了後、gss_release_buffer() を使って解放する必要があります。

input-name-type *input-name-buffer* の形式を示す gss_OID。71 ページの「GSS-API における名前型」を参照してください。また、235 ページの「名前型」に、有効な名前型の一覧表があります。

output-name 名前を受け取る gss_name_t 構造体。

次に示すのは、例 4–1 の汎用例に若干の変更を加えたものであり、gss_import_name() の使用法を示しています。まず、通常の文字列が gss_buffer_desc 構造体に挿入されています。次に、その文字列が、gss_import_name() によって gss_name_t 構造体内に格納されています。

例 4–2 gss_import_name()() の使用例

```
char *name_string;
gss_buffer_desc input_name_buffer;
gss_name_t      output_name_buffer;

input_name_buffer.value = name_string;
input_name_buffer.length = strlen(input_name_buffer.value) + 1;

gss_import_name(&minor_status, input_name_buffer,
                    GSS_C_NT_HOSTBASED_SERVICE, &output_name);

gss_release_buffer(input_name_buffer);
```

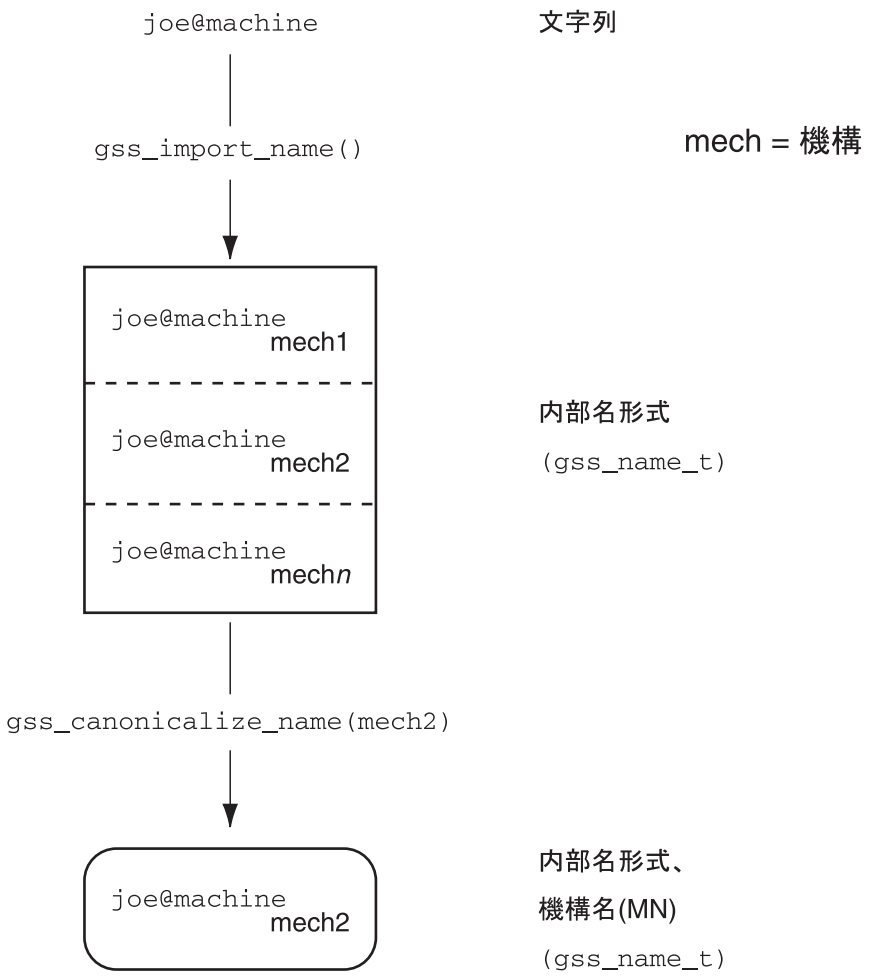
インポートされた名前は、人間が読める形式で表示できるように、元の gss_buffer_t オブジェクトに戻すことが可能です。それには、gss_display_name() を使用します。ただし、gss_display_name() は、結果の文字列が元と同じであることを保証しません。その原因は、実際の機構が名前を格納する方法にあります。GSS-API にはほかにも名前を処理する関数があります。227 ページの「GSS-API 関数」を参照してください。

gss_name_t 構造体には、単一の名前の複数のバージョンを格納できます。GSS-API によってサポートされている機構ごとに、1 つのバージョンが生成されます。つまり、*user@company* の gss_name_t 構造体には、Kerberos v5 から提供されたその名前の

1 つのバージョンと、別の機構から提供された別のバージョンが含まれる可能性があります。関数 gss_canonicalize_name() は、入力として内部名と機構を受け取ります。また、gss_canonicalize_name() は、出力としてその機構に固有の単一バージョン名だけを含む別の内部名を返します。

そうした機構に固有な名前のことを「機構名 (Mechanism Name、MN)」と呼びます。機構名とは、特定の機構の名前ではなく、特定の機構によって生成された主体の名前です。このプロセスを示したのが次の図です。

図 4–3　内部名と機構名 (MN)



GSS-API における名前の比較

サーバーがクライアントからある名前を受け取り、その名前をアクセス制御リスト内で検索する必要がある場合を考えます。「アクセス制御リスト (Access Control List、ACL)」とは、特定のアクセス権を持つ主体のリストのことです。そうした検索を行うには、次のような方法が考えられます。

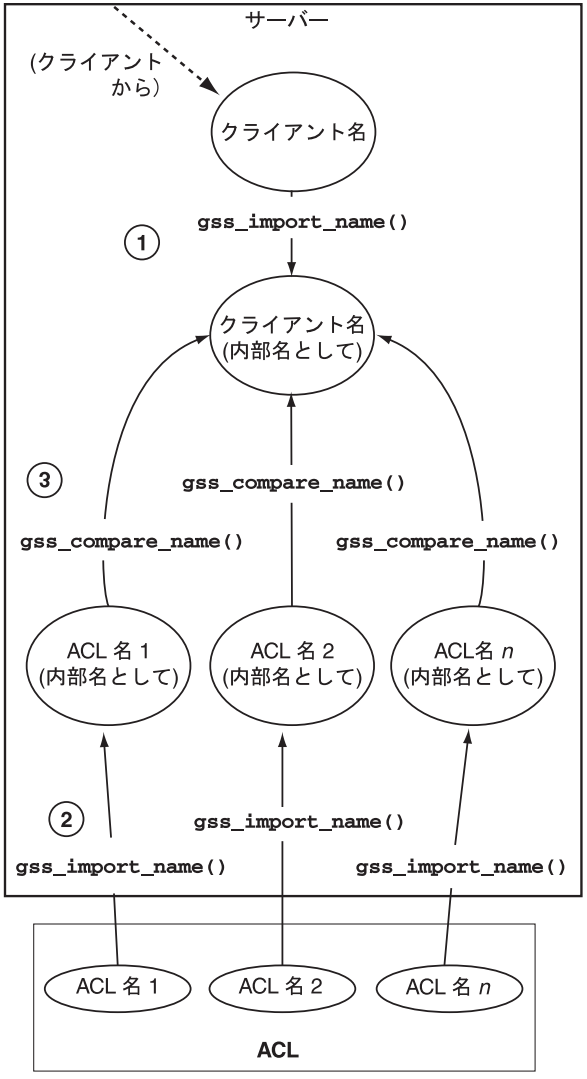
1. gss_import_name() で、クライアント名を GSS-API 内部形式にインポートします (まだインポートされていない場合)。

   場合によっては、サーバーは名前を内部形式で受け取ります。その場合、この手順は必要ありません。たとえば、サーバーはクライアント自身の名前を検索する可能性があります。コンテキストの起動中、クライアント自身の名前は内部形式で渡されます。

2. gss_import_name() で、各 ACL 名を インポートします。
3. gss_compare_name() で、インポートした各 ACL 名をインポートしたクライアント名と比較します。

次の図に、このプロセスを示します。ここでは手順 1 が必要であると仮定します。

図 4–4　名前の比較 (遅い)



名前の数が少ない場合は、名前を個別に比較する上記の方法でも問題ありません。名前の数が非常に多い場合は、`gss_canonicalize_name()` 関数を使用するほうが効率的です。この方法の実行手順を次に示します。

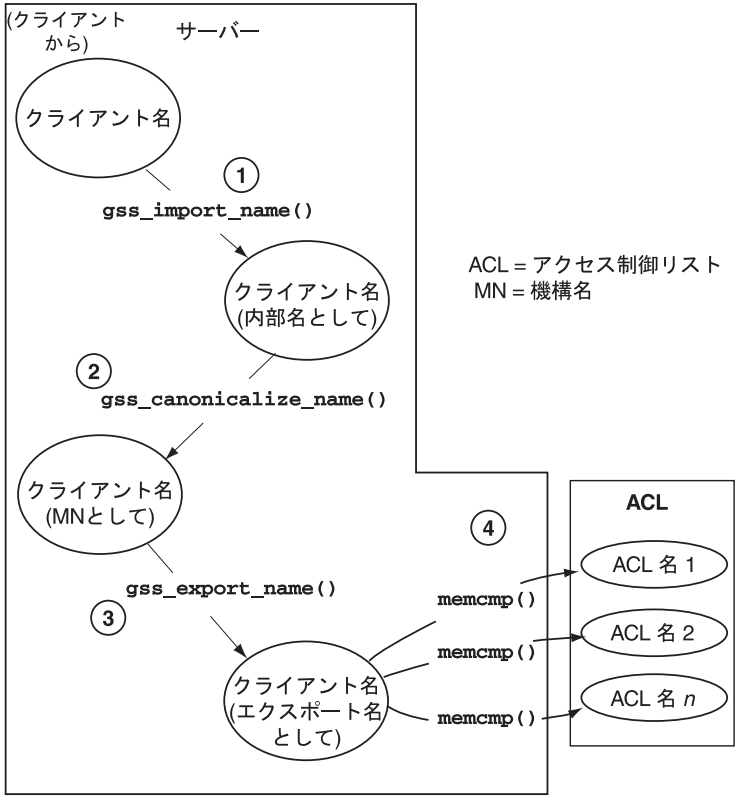
1. `gss_import_name()` で、クライアント名をインポートします (まだインポートされていない場合)。

   名前を比較する前述の方法と同様に、名前がすでに内部形式である場合には、この手順は必要ありません。

2. gss_canonicalize_name() を使用してクライアント名の機構名バージョンを生成します。
3. gss_export_name() を使用してエクスポート名を生成します。エクスポート名は、連続した文字列としてのクライアント名です。
4. memcmp() を使用してエクスポートされたクライアント名を ACL 内の個々の名前と比較します。これは、高速で動作するオーバーヘッドの少ない関数です。

次の図に、このプロセスを示します。ここでも、サーバーがクライアントから受信した名前をインポートする必要があると仮定します。

図 4–5 名前の比較 (速い)



gss_export_name() は機構名 (MN) を期待するため、先にクライアント名に対して gss_canonicalize_name() を実行する必要があります。

詳細は、gss_export_name(3GSS)、gss_import_name(3GSS)、および gss_canonicalize_name(3GSS) を参照してください。

GSS-API の OID

オブジェクト識別子 (Object Identifier、OID) は、次のようなデータを格納するときに使用します。

- セキュリティー機構
- QOP – Quality of Protection (保護品質) の値
- 名前型

OID は、GSS-API の `gss_OID_desc` 構造体に格納されます。次の例のように、GSS-API はその構造体へのポインタ `gss_OID` を提供します。

例 4–3 OID の構造体

```
typedef struct gss_OID_desc_struct {
        OM_uint32   length;
        void        *elements;
     } gss_OID_desc, *gss_OID;
```

さらに、1 つ以上の OID を `gss_OID_set_desc` 構造体に格納することもできます。

例 4–4 OID セットの構造体

```
typedef struct gss_OID_set_desc_struct {
        size_t    count;
        gss_OID   elements;
     } gss_OID_set_desc, *gss_OID_set;
```

注意 – アプリケーションは `free()` で OID を解放するべきではありません。

GSS-API における機構と QOP

GSS-API では、使用するセキュリティー機構をアプリケーションが選択できるようになっていますが、GSS-API が選択したデフォルトの機構をできる限り使用する必要があります。同様に、GSS-API では、データ保護の保護品質レベルをアプリケーションが指定できるようになっていますが、デフォルトの QOP をできる限り使用する必要があります。デフォルトの機構を受け入れることを示すには、機構または QOP を期待する関数に値 `GSS_C_NULL_OID` を引数として渡します。

注意 – セキュリティー機構またはQOPを明示的に指定することは、GSS-APIの使用目的に反します。そうした特定の選択は、アプリケーションの移植性を制限します。ほかのGSS-API実装は、そのQOPまたは機構を意図した方法でサポートしていない可能性があります。ただし、付録C「OIDの指定」では、利用可能な機構やQOPを知る方法と、それらの選択方法について、簡単に説明しています。

GSS-APIにおける名前型

QOPとセキュリティー機構のほかに、名前型を示すためにもOIDが使用されます。名前型とは、関連する名前の形式を示すものです。たとえば、`gss_import_name()`関数は主体の名前を文字列から`gss_name_t`型に変換しますが、この関数は変換すべき文字列の形式を引数の1つとして受け取ります。たとえば、名前型が`GSS_C_NT_HOSTBASED_SERVICE`である場合、この関数は、入力された名前が*service@host*形式であると判断します。名前型が`GSS_C_NT_EXPORT_NAME`である場合、この関数はGSS-APIエクスポート名を期待します。アプリケーションは`gss_inquire_names_for_mech()`関数を使用すると、指定した機構で使用できる名前型を知ることができます。GSS-APIによって使用される名前型の一覧については、235ページの「名前型」を参照してください。

GSS-API状態コード

すべてのGSS-API関数は、関数の成功または失敗に関する情報を提供する2種類のコードを返します。どちらの種類の状態コードも`OM_uint32`値として戻されます。次に、この2種類の戻りコードについて説明します。

- メジャー状態コード – 次のエラー状態を示すコードです。
  - 汎用GSS-APIルーチンエラー (ルーチンに無効な機構を指定したなど)
  - 特定のGSS-API言語バインディングに固有の呼び出しエラー (関数の引数が読み書きできない、引数の形式が間違っているなど)
  - 両方のタイプのエラー

  さらに、メジャー状態コードは、ルーチンの状態に関する補足情報も提供できます。たとえば、処理が終了していない、トークンの送信順が間違っている、などを示すコードが返されます。何もエラーが発生しなかった場合、ルーチンは値が`GSS_S_COMPLETE`のメジャー状態コードを戻します。

  メジャー状態コードは次のようにして返されます。

```
OM_uint32 major_status ;    /* status returned by GSS-API */

major_status = gss_generic_function(arg1, arg2 ...);
```

メジャー状態戻りコードは他の OM_uint32 と同じように処理できます。たとえば、次のコードを考えます。

```
OM_uint32 maj_stat;

maj_sta = gss_generic_function(arg1, arg2 ...);

if (maj_stat == GSS_CREDENTIALS_EXPIRED)
        <do something...>
```

メジャー状態コードは、マクロ GSS_ROUTINE_ERROR()、GSS_CALLING_ERROR()、および GSS_SUPPLEMENTARY_INFO() で処理できます。230 ページの「GSS-API 状態コード」では、メジャー状態コードの読み取り方法について説明しているほか、GSS-API 状態コードの一覧を提供しています。

- マイナー状態コード – 実際の機構から返されるコード。これらのコードについては、このマニュアルでは具体的に説明しません。

  すべての GSS-API 関数は最初の引数として OM_uint32 型のマイナー状態コードを受け取ります。関数が呼び出し元の関数に制御を戻す際に、その OM_uint32 引数にマイナー状態コードが格納されます。次のコードを考えます。

  ```
  OM_uint32 *minor_status ;    /* status returned by mech */

  major_status = gss_generic_function(&minor_status, arg1, arg2 ...);
  ```

  致命的なエラーを示すメジャー状態コードが返される場合でも、*minor_status* パラメータは GSS-API ルーチンによって必ず設定されます。その他のほとんどの出力パラメータには値が設定されません。ただし、ルーチンによって割り当てられた記憶領域へのポインタを返すべき出力パラメータには、NULL が設定されます。NULL は、記憶領域が実際には割り当てられなかったことを示します。このようなポインタに関連する長さフィールド (gss_buffer_desc 構造体を参照) は 0 に設定されます。そのような場合、アプリケーションはこれらのバッファーを解放する必要はありません。

GSS-API トークン

GSS-API における「流通」の基本単位は「トークン」です。GSS-API を使用するアプリケーションは、トークンを使用して互いに通信します。トークンは、データを交換したりセキュリティーを確立したりするために使われます。トークンは gss_buffer_t データ型として宣言されます。トークンはアプリケーションに対して不透明です。

トークンには、「コンテキストレベルトークン」と「メッセージ毎トークン」の 2 種類があります。コンテキストレベルトークンは主に、コンテキストを確立する際、つまりコンテキストを起動して受け入れる際に使用されます。コンテキストレベルトークンは、コンテキストを管理する目的で、後になって渡されることがあります。

メッセージ毎トークンは、コンテキストが確立されたあとで使用されます。メッセージ毎トークンは、データ保護サービスを提供する目的で使用されます。たとえば、別のアプリケーションにメッセージを送信したいアプリケーションを考えます。そのアプリケーションは、GSS-API を使って暗号化識別子を生成し、それをメッセージに添付します。その識別子はトークンに格納されます。

メッセージ毎トークンは、メッセージとの関係において次のように考えることができます。「メッセージ」とは、アプリケーションがピアに送信するデータです。たとえば、`ls` コマンドは、`ftp` サーバーに送信されるメッセージになりえます。メッセージ毎トークンとは、そのメッセージに対して GSS-API が生成するオブジェクトのことです。メッセージ毎トークンの例としては、暗号タグや暗号化された形式のメッセージが挙げられます。ただし、後者の例は若干不正確です。暗号化されたメッセージはやはりメッセージであり、トークンではありません。トークンと呼べるのは、GSS-API によって生成された情報だけです。 しかし、正式にではありませんが、「メッセージ」と「メッセージ毎トークン」は同じ意味で使用されることがあります。

次の作業はアプリケーションの責任です。

1. トークンを送受信すること。開発者はこのようなアクションを実行するために、通常、汎用的な読み取り関数と書き込み関数を作成する必要があります。219 ページの「その他の GSS-API 関数例」の `send_token()` 関数と `recv_token()` 関数を参照してください。
2. トークンの種類を区別し、それに応じてトークンを操作すること。

   トークンはアプリケーションに対して不透明であるため、アプリケーションは、あるトークンと別のトークンを区別できません。トークンの内容がわからなくても、アプリケーションはトークンの種類を区別できる必要があります。なぜなら、そうしないとトークンを適切な GSS-API 関数に渡せないからです。アプリケーションは、次の方法でトークンの種類を区別できます。

   - 状態によって (プログラムの制御フローを通じて)。たとえば、コンテキストを受け入れるために待機しているアプリケーションは、受信したトークンはコンテキストの確立に関係するものであると仮定します。ピアは、コンテキストが完全に確立されるまで、メッセージトークン (つまりデータ) の送信を行わないと予想されます。いったんコンテキストが確立されると、アプリケーションは新しいトークンがメッセージトークンであると仮定します。このようなトークンの処理方法は、非常に一般的なものです。後述のプログラム例でもこの方法を使用しています。
   - フラグによって。たとえば、トークンをピアに送信するための関数がアプリケーションに含まれている場合、そのアプリケーションにはトークンの種類を示すフラグを含めることができます。次のコードを考えます。

     ```
     gss_buffer_t token;     /* declare the token */
     OM_uint32 token_flag       /* flag for describing the type of token */
     ```

```
<get token from a GSS-API function>

token_flag = MIC_TOKEN;     /* specify what kind of token it is */
send_a_token(&token, token_flag);
```

受信側のアプリケーションは、受信関数 (`get_a_token()` など) で *token_flag* 引数を検査します。

- 明示的なタグ付けによって。アプリケーションは「メタトークン」を使用できます。メタトークンは、GSS-API 関数から受け取ったトークンを格納するためのユーザー定義の構造体です。メタトークンには、GSS-API から提供されたトークンの使用方法を示すユーザー定義フィールドが含まれます。

GSS-API におけるプロセス間トークン

GSS-API では、マルチプロセスアプリケーション内のあるプロセスから別のプロセスにセキュリティーコンテキストを渡せます。通常、アプリケーションはクライアントのコンテキストを受け入れます。アプリケーションはそのコンテキストをアプリケーション内のプロセス間で共有します。マルチプロセスアプリケーションについては、85 ページの「GSS-API におけるコンテキストのエクスポートとインポート」を参照してください。

`gss_export_context()` 関数はプロセス間トークンを作成します。このトークンに含まれる情報を使えば、2 番目のプロセス内でコンテキストを再構築できます。あるプロセスから別のプロセスにプロセス間トークンを渡すのは、アプリケーションの責任です。この状況は、トークンを別のアプリケーションに渡すのがアプリケーションの責任であることに似ています。

プロセス間トークンには、鍵などの機密情報が含まれる可能性があります。必ずしもすべての GSS-API 実装がプロセス間トークンを暗号技術で保護するとは限りません。したがって、アプリケーションは、プロセス間トークンに保護を施したあとで交換を実施する必要があります。そうした保護は、`gss_wrap()` でトークンを暗号化するなどして実現します (ただし、暗号化が利用可能である場合)。

注 – 異なる GSS-API 実装間では、プロセス間トークンを転送できるとは限りません。

GSS-API を使用するアプリケーションの開発

ここでは、GSS-API を使用してセキュリティー保護されたデータ交換処理を実装する方法について説明します。ただし、GSS-API を使用するうえでもっとも中心となる関数に焦点を当てます。詳細は、付録 B 「GSS-API リファレンス」を参照してくださ

い。この付録には、すべての GSS-API 関数、状態コード、およびデータ型の一覧が含まれています。各 GSS-API 関数についての詳細は、個々のマニュアルページを参照してください。

このマニュアルの例では単純なモデルを使用します。クライアントアプリケーションがリモートサーバーに直接データを送信します。RPC などのトランスポートプロトコル層による仲介は発生しません。

GSS-API の一般的な使用法

GSS-API を使用する際の一般的な手順は、次のとおりです。

1. 各アプリケーション (送信側と受信側の両方) は資格を明示的に獲得します (資格を自動的に獲得していない場合)。
2. 送信側はセキュリティーコンテキストを起動します。受信側はそのコンテキストを受け入れます。
3. 送信側は転送するデータにセキュリティー保護を適用します。送信側はメッセージを暗号化するか、データに識別タグを付けます。その後、送信側は保護されたメッセージを転送します。

   注 – 送信側はセキュリティー保護を適用しなくてもかまいません。その場合、デフォルトの GSS-API セキュリティーサービスである認証だけがメッセージに適用されます。

4. 受信側はメッセージを復号化し (必要であれば)、メッセージを検証します (該当する場合)。
5. (省略可能) 確認のため、受信側は識別タグを送信側に返送します。
6. 送信側と受信側のアプリケーションは両方とも共有セキュリティーコンテキストを無効にします。必要であれば、アプリケーションは残りの GSS-API データもすべて解放します。

注意 – 割り当てられたすべてのデータ領域を解放することは、呼び出し元のアプリケーションの責任です。

GSS-API を使用するアプリケーションは、ファイル `gssapi.h` をインクルード (`include`) する必要があります。

GSS-API における資格の操作

「資格」とは、主体名に対するアプリケーションの要求の証明を提供するデータ構造です。アプリケーションは、資格を使って自身の大域アイデンティティーを確立します。さらに資格は、エンティティーの権限を確認する目的で使用される場合もあります。

GSS-API 自身は資格を提供しません。資格は、GSS-API 関数が呼び出される前に、GSS-API の背後にあるセキュリティー機構によって作成されます。多くの場合、ユーザーはログイン時に資格を受け取ります。

ある特定の GSS-API 資格は単一の主体に対してのみ有効です。単一の資格には、その主体に対する要素が複数個含まれる可能性があります。それらの要素は機構ごとに 1 つずつ作成されます。複数のセキュリティー機構を備えたマシン上で獲得した資格は、それらの機構のサブセットを備えたマシンに転送された場合に有効になります。GSS-API は `gss_cred_id_t` 構造体を通じて資格にアクセスします。この構造体のことを「資格ハンドル」と呼びます。資格はアプリケーションに対して不透明です。したがって、アプリケーションは与えられた資格の詳細を知る必要はありません。

資格には 3 つの形式があります。

- `GSS_C_INITIATE` – セキュリティーコンテキストの起動のみを行うアプリケーションを識別します
- `GSS_C_ACCEPT` – セキュリティーコンテキストの受け入れのみを行うアプリケーションを識別します
- `GSS_C_BOTH` – セキュリティーコンテキストの起動と受け入れを行えるアプリケーションを起動します

GSS-API における資格の獲得

セキュリティーコンテキストが確立できるようになるまでに、サーバーとクライアントはそれぞれの資格を獲得する必要があります。資格は有効期限が切れるまで何度でも使用できます。有効期限が切れると、アプリケーションは資格を獲得し直す必要があります。クライアントが使用する資格とサーバーが使用する資格とでは、その有効期間が異なる場合があります。

GSS-API ベースのアプリケーションが資格を獲得する方法には、次の 2 つがあります。

- `gss_acquire_cred()` または `gss_add_cred()` 関数を使用する方法
- コンテキスト確立時にデフォルトの資格を示す値 `GSS_C_NO_CREDENTIAL` を指定する方法

ほとんどの場合、gss_acquire_cred() を呼び出すのは、コンテキストの受け入れ側、つまりサーバーだけです。コンテキストの起動側、つまりクライアントは一般に、ログイン時に資格を受け取ります。したがって、クライアントは通常、デフォルトの資格を指定できます。サーバーは、gss_acquire_cred() を使用せずに、自身のデフォルトの資格を使用することもできます。

クライアントの資格は、そのクライアントの身元をほかのプロセスに対して証明します。サーバーが資格を獲得すると、セキュリティーコンテキストの受け入れが可能になります。したがって、クライアントがサーバーに ftp 要求を送る場合、そのクライアントはログイン時からすでに資格を獲得している可能性があります。クライアントがコンテキストを起動しようとすると、GSS-API は自動的にその資格を取得します。しかし、サーバープログラムは要求されたサービス (ftp) の資格を明示的に獲得します。

gss_acquire_cred() が正常終了すると、GSS_S_COMPLETE が返されます。有効な資格を返せない場合は、GSS_S_NO_CRED が返されます。その他のエラーコードについては、gss_acquire_cred(3GSS) のマニュアルページを参照してください。使用例については、第 6 章の「資格の獲得」を参照してください。

gss_add_cred() は gss_acquire_cred() に似ています。しかし、gss_add_cred() を使用すれば、アプリケーションは既存の資格を基に新しいハンドルを作成したり、既存の資格に新しい資格要素を追加したりできます。GSS_C_NO_CREDENTIAL を既存の資格として指定した場合、gss_add_cred() はデフォルトの動作に従って新しい資格を作成します。詳細は、gss_add_cred(3GSS) のマニュアルページを参照してください。

GSS-API におけるコンテキストの操作

GSS-API がセキュリティー提供時に行うもっとも重要な作業は、セキュリティーコンテキストを作成することと、データを保護することの 2 つです。アプリケーションは必要な資格を獲得したあと、セキュリティーコンテキストを確立する必要があります。コンテキストを確立するには、一方のアプリケーション (通常はクライアント) がコンテキストを起動し、もう一方のアプリケーション (通常はサーバー) がそのコンテキストを受け入れます。ピア間で複数のコンテキストが存在してもかまいません。

通信中のアプリケーションは、認証トークンを交換することによって、結合セキュリティーコンテキストを確立します。セキュリティーコンテキストは、2 つのアプリケーション間で共有すべき情報が入っている一対の GSS-API データ構造体です。この情報は、各アプリケーションのセキュリティーにおける状態を記述します。セキュリティーコンテキストはデータの保護のために必要です。

GSS-API におけるコンテキストの起動

アプリケーションとリモートピア間でセキュリティーコンテキストを起動するには、gss_init_sec_context() 関数が使用されます。処理が成功すると、この関数は、確立すべきコンテキストの「コンテキストハンドル」と、受け入れ側に送信すべきコンテキストレベルトークンを返します。gss_init_sec_context() を呼び出す前に、クライアントは次の作業を行う必要があります。

1. 必要であれば、gss_acquire_cred() で資格を獲得します。通常の場合、クライアントはログイン時に資格を受け取ります。gss_acquire_cred() は単純に、実行中のオペレーティングシステムから初期の資格を取得できます。
2. gss_import_name() で、サーバー名を GSS-API 内部形式にインポートします。名前と gss_import_name() についての詳細は、63 ページの「GSS-API における名前」を参照してください。

gss_init_sec_context() を呼び出す際、クライアントは通常、次の引数値を渡します。

- GSS_C_NO_CREDENTIAL を引数 *cred_handle* に渡して、デフォルトの資格を示します
- GSS_C_NULL_OID を引数 *mech_type* に渡して、デフォルトの機構を示します
- GSS_C_NO_CONTEXT を引数 *context_handle* に渡して、初期コンテキストが空であることを示します。gss_init_sec_context() は通常ループ内で呼び出されるため、後続の呼び出しは以前の呼び出しで戻されたコンテキストハンドルを渡す必要があります
- GSS_C_NO_BUFFER を引数 *input_token* に渡して、トークンが最初は空であることを示します。あるいは、アプリケーションは length フィールドが 0 に設定されている gss_buffer_desc オブジェクトへのポインタを渡すこともできます
- gss_import_name() で GSS-API 内部形式にインポートされたサーバー名を渡します

アプリケーションは必ずしもこのようなデフォルト値を使用する必要はありません。さらに、クライアントは引数 *req_flags* を使用して、他のセキュリティーパラメータに対する要件を指定することもできます。gss_init_sec_context() の引数についての詳細は、以降の節で説明します。

コンテキスト受け入れ側は、コンテキスト確立時にいくつかのハンドシェークを要求する可能性があります。つまり、受け入れ側は、コンテキストが完全に確立されるまで、複数のコンテキスト情報を送信するように起動側に要求できます。したがって、移植性のため、コンテキストの起動は常に、コンテキストが完全に確立されたかどうかを検査するループの一部として行われる必要があります。

コンテキストが完全に確立されていない場合、gss_init_sec_context() はメジャー状態コードとして GSS_C_CONTINUE_NEEDED を戻します。したがって、その gss_init_sec_context() からの戻り値を使用して、起動ループを継続するかどうかの判定を行う必要があります。

クライアントはコンテキスト情報をサーバーに、gss_init_sec_context() から戻された「出力トークン」の形式で渡します。クライアントは、サーバーから情報を「入力トークン」として受け取ります。その後、その入力トークンは、後続の gss_init_sec_context() 呼び出しの引数として渡すことができます。受け取った入力トークンの長さが 0 の場合、サーバーはこれ以上出力トークンを要求していないことが分かります。

したがって、ループ内で gss_init_sec_context() の戻り値の状態を検査する以外に、入力トークンの長さも検査する必要があります。その長さが 0 以外の値である場合、別のトークンをサーバーに送信する必要があります。ループを開始する前に、入力トークンの長さを 0 に初期化しておく必要があります。入力トークンを GSS_C_NO_BUFFER に設定するか、その構造体の length フィールドの値を 0 に設定します。

次の擬似コードは、クライアント側からのコンテキストの確立例を示したものです。

```
コンテキストを GSS_C_NO_CONTEXT で初期化する
入力トークンを GSS_C_NO_BUFFER で初期化する

do

   call gss_init_sec_context(資格, コンテキスト, 名前, 入力トークン,
                             出力トークン, その他の引数...)

   if (受け入れ側に送信すべき出力トークンが存在する)
      受け入れ側に出力トークンを送信する
      出力トークンを解放する

   if (コンテキストが完全でない)
      受け入れ側から入力トークンを受信する

   if (GSS-API エラーが発生した)
      コンテキストを削除する

until コンテキストが完成
```

実際のループは、さまざまなエラー検査を含んだ、より複雑なものになります。そうしたコンテキスト起動ループの実際の例については、103 ページの「サーバーとのセキュリティーコンテキストの確立」を参照してください。さらに、gss_init_sec_context(3GSS) のマニュアルページにも、上記例ほど一般化されていない例があります。

一般に、コンテキストが完全に確立されていない時に戻されるパラメータ値は、コンテキストが完了した時に戻されるはずの値です。詳細は、gss_init_sec_context() のマニュアルページを参照してください。

gss_init_sec_context() が正常終了すると、GSS_S_COMPLETE が返されます。コンテキスト確立トークンがピアとなるアプリケーションから要求された場合、GSS_S_CONTINUE_NEEDED が返されます。エラーが発生した場合、gss_init_sec_context(3GSS) のマニュアルページに記載されたエラーコードが返されます。

コンテキストの起動が失敗した場合、クライアントはサーバーから切断する必要があります。

GSS-API におけるコンテキストの受け入れ

コンテキストの確立におけるもう 1 つの仕事は、コンテキストの受け入れです。コンテキストの受け入れは gss_accept_sec_context() 関数で行います。通常の場合、クライアントが gss_init_sec_context() で起動したコンテキストを、サーバーが受け入れます。

gss_accept_sec_context() への主な入力は、起動側から受け取った入力トークンです。gss_accept_sec_context は、コンテキストハンドルと起動側に戻すべき出力トークンを戻します。しかし、gss_accept_sec_context() を呼び出す前に、サーバーはクライアントから要求されたサービスの資格を獲得しておく必要があります。サーバーはこのような資格を gss_acquire_cred() 関数で獲得します。あるいは、サーバーは、資格を明示的に獲得するのではなく、gss_accept_sec_context() を呼び出す際にデフォルトの資格 (GSS_C_NO_CREDENTIAL) を指定することもできます。

gss_accept_sec_context() を呼び出すとき、サーバーは次の引数を設定できます。

- *cred_handle* – gss_acquire_cred() によって返された資格ハンドル。あるいは、デフォルトの資格を示す GSS_C_NO_CREDENTIAL も使用できます。
- *context_handle* – GSS_C_NO_CONTEXT は初期コンテキストが空であることを示します。gss_init_sec_context() は通常ループ内で呼び出されるため、後続の呼び出しは以前の呼び出しで戻されたコンテキストハンドルを渡す必要があります。
- *input_token* – クライアントから受け取ったコンテキストトークン。

gss_accept_sec_context() 引数についての詳細は、以降の節で説明します。

セキュリティーコンテキストを確立するためには、いくつかのハンドシェークが必要となる可能性があります。コンテキストが完全に確立されるまでに、起動側と受け入れ側は通常、複数のコンテキスト情報を送信する必要があります。したがって、移植性のため、コンテキストの受け入れは常に、コンテキストが完全に確立されたかどうかを検査するループの一部として行われる必要があります。コンテキストがまだ確立されていない場合、gss_accept_sec_context() はメジャー状態

コード GSS_C_CONTINUE_NEEDED を返します。したがって、ループは gss_accept_sec_context() の戻り値を使用して、受け入れループを継続するかどうかを判定する必要があります。

コンテキスト受け入れ側はコンテキスト情報をコンテキスト起動側に、gss_accept_sec_context() から戻された出力トークンの形式で渡します。その後、受け入れ側は、起動側から追加情報を入力トークンとして受け取れます。入力トークンは、後続の gss_accept_sec_context() 呼び出しの引数として渡されます。起動側に送信すべきトークンがなくなると、gss_accept_sec_context() から長さ 0 の出力トークンが返されます。ループ内で、gss_accept_sec_context() の戻り値の状態を検査する以外に、出力トークンの長さを検査して、別のトークンを送信すべきかどうかを判断する必要があります。ループを開始する前に、出力トークンの長さを 0 に初期化しておく必要があります。出力トークンを GSS_C_NO_BUFFER に設定するか、その構造体の length フィールドの値を 0 に設定します。

次の擬似コードは、サーバー側からのコンテキストの確立例を示したものです。

```
コンテキストを GSS_C_NO_CONTEXT で初期化する
出力トークンを GSS_C_NO_BUFFER で初期化する

do

   起動側から入力トークンを受信する

   call gss_accept_sec_context(コンテキスト, 資格ハンドル, 入力トークン,
                               出力トークン, その他の引数...)

   if (起動側に送信すべき出力トークンが存在する)
      起動側に出力トークンを送信する
      出力トークンを解放する

   if (GSS-API エラーが発生した)
      コンテキストを削除する

until コンテキストが完成
```

実際のループは、さまざまなエラー検査を含んだ、より複雑なものになります。そうしたコンテキスト受け入れループの実際の例については、103 ページの「サーバーとのセキュリティーコンテキストの確立」を参照してください。さらに、gss_accept_sec_context() のマニュアルページにも例が記載されています。

繰り返しになりますが、GSS-API 自身はトークンを送受信しません。トークンの送受信はアプリケーションが処理する必要があります。トークンを転送する関数の例については、219 ページの「その他の GSS-API 関数例」を参照してください。

正常に終了した場合、gss_accept_sec_context() は GSS_S_COMPLETE を戻します。コンテキストが完全に確立されていない場合、gss_accept_sec_context は GSS_S_CONTINUE_NEEDED を戻します。エラーが発生した場合、この関数はエラーコードを戻します。詳細は、gss_accept_sec_context(3GSS) のマニュアルページを参照してください。

GSS-API におけるその他のコンテキストサービスの使用

gss_init_sec_context() 関数では、アプリケーションは、基本的なコンテキストの確立以外に、追加のデータ保護サービスを要求できるようになっています。このようなサービスを要求するには、gss_init_sec_context() の *req_flags* 引数を使用します。

すべての機構がこれらすべてのサービスを提供するわけではありません。gss_init_sec_context() の *ret_flags* 引数は、指定されたコンテキストでどのサービスが利用できるかを示します。同様に、コンテキスト受け入れ側では、gss_accept_sec_context() が返す *ret_flags* 値から、どのサービスを利用できるかを判断します。以降の節では、追加のサービスについて説明します。

GSS-API における資格の委託

許可されていれば、コンテキスト起動側はコンテキスト受け入れ側が代理として動作するように要求できます。そのような場合、受け入れ側は、起動側に代わって別のコンテキストを起動できます。

マシン A 上のあるユーザーが、マシン B に rlogin したあと、さらにマシン B からマシン C に rlogin したいものとします。このとき、委託された資格は B を A として識別するか、あるいは A のプロキシとして動作している B として識別するかは、機構によって異なります。

委託が許可されると、*ret_flags* に値 GSS_C_DELEG_FLAG を設定できます。受け入れ側は委託された資格を gss_accept_sec_context() の *delegated_cred_handle* 引数として受け取ります。資格の委託はコンテキストのエクスポートとは異なります。85 ページの「GSS-API におけるコンテキストのエクスポートとインポート」を参照してください。その違いの 1 つは、アプリケーションの資格は一度に複数回委託できますが、コンテキストは一度に 1 つのプロセスでしか保持できない、という点です。

GSS-API におけるピア間での相互認証の実行

ftp サイトにファイルを転送するユーザーは通常、そのサイトの身元についての証明を必要としません。これに対し、アプリケーションからクレジットカード番号の提供を求められたユーザーは、その受信側の身元についての確実な証明を得たいはずです。そうした場合に「相互認証」が必要になります。コンテキストの起動側と受け入れ側の両方が、自身の身元を証明する必要があります。

コンテキスト起動側が相互認証を要求するには、gss_init_sec_context() の *req_flags* 引数に値 GSS_C_MUTUAL_FLAG を設定します。相互認証が承認されると、この関数はそのことを示すために、*ret_flags* 引数にこの値を設定します。相互認証が要求されたが使用できない場合、適切な対処を行うのは起動側のアプリケーションの責任です。相互認証が要求されたが使用できない場合でも、GSS-API は自動的にコンテキストを終了しません。また、機構の中には、特に要求がなくても相互認証を常に実行するものもあります。

GSS-API における匿名認証の実行

GSS-API の通常の使用においては、起動側の識別情報はコンテキスト確立の一部として、受け入れ側で使用できるようになります。しかし、コンテキスト起動側は自身の識別情報をコンテキスト受け入れ側に知らせないように要求することもできます。

たとえば、医療データベースへの無制限のアクセスを提供するアプリケーションを考えます。そのようなサービスのクライアントは、サービスの認証を要求することが考えられます。この方法では、データベースから取り出されるすべての情報に対して信頼が確立されます。しかし、クライアントはプライバシー上の理由などにより、識別情報を公開したくないかもしれません。

匿名性を要求するには、gss_init_sec_context() の *req_flags* 引数に GSS_C_ANON_FLAG を設定します。匿名性が利用可能かどうかを検査するには、gss_init_sec_context() または gss_accept_sec_context() の *ret_flags* 引数に GSS_C_ANON_FLAG が返されるかどうかを確認します。

匿名性が有効である場合、gss_accept_sec_context() または gss_inquire_context() から返されたクライアント名を指定して gss_display_name()を呼び出すと、汎用的な匿名が生成されます。

注 – 匿名性が要求されたが使用できない場合、適切な対処を行うのはアプリケーションの責任です。GSS-API はそのような場合にコンテキストを終了しません。

GSS-API におけるチャネルバインディングの使用

多くのアプリケーションでは、基本的なコンテキスト確立を行うだけで、コンテキスト起動側を適切に認証できます。追加のセキュリティーが必要な場合、GSS-API ではチャネルバインディングを使用します。チャネルバインディングとは、使用されている特定のデータチャネルを識別するためのタグのことです。具体的には、チャネルバインディングはコンテキストの起点と終点 (つまり起動側と受け入れ側) を識別します。これらのタグは起動側と受け入れ側のアプリケーションに固有であるため、識別情報のより有効な証明となります。

チャネルバインディングは、次に示すように、gss_channel_bindings_struct 構造体へのポインタである gss_channel_bindings_t データ型によって示されます。

```
typedef struct gss_channel_bindings_struct {
OM_uint32       initiator_addrtype;
gss_buffer_desc initiator_address;
OM_uint32       acceptor_addrtype;
gss_buffer_desc acceptor_address;
gss_buffer_desc application_data;
} *gss_channel_bindings_t;
```

最初の 2 つのフィールドは起動側のアドレスとアドレス型 (起動側のアドレスが送信される形式) を示します。たとえば、*initiator_addrtype* を GSS_C_AF_INET に設定した場合、*initiator_address* がインターネットアドレス形式 (つまり IP アドレス) であることを示します。同様に、3 番目と 4 番目のフィールドは受け入れ側のアドレスとアドレス型を示します。最後のフィールド *application_data* は、アプリケーションが自由に使用することができます。*application_data* を使用する予定がない場合、このフィールドを GSS_C_NO_BUFFER に設定します。アプリケーションがアドレスを指定しない場合、アドレス型フィールドを GSS_C_AF_NULLADDR に設定します。有効なアドレス型の値については、236 ページの「チャネルバインディングのアドレス型」を参照してください。

アドレス型は、特定のアドレス形式を示すのではなく、アドレスファミリを示します。アドレスファミリが複数の代替アドレス形式を持つ場合、どの形式を使用しているかを判断できるだけの十分な情報を、*initiator_address* と *acceptor_address* のフィールドに指定する必要があります。特に指定しない限り、アドレスはネットワークのバイト順 (つまり、アドレスファミリにネイティブなバイト順) で指定します。

チャネルバインディングを使用してコンテキストを確立するには、割り当てられたチャネルバインディング構造体を gss_init_sec_context() の *input_chan_bindings* 引数で指します。この構造体の各フィールドが連結されてオクテット文字列が生成され、そこから MIC が派生されます。次に、この MIC が出力トークンに添付されます。続いて、アプリケーションはそのトークンをコンテキスト受け入れ側に送信します。トークンを受け取った受け入れ側は、gss_accept_sec_context() を呼び出します。詳細は、80 ページの「GSS-API におけるコンテキストの受け入れ」を参照してください。gss_accept_sec_context() は、受け取ったチャネルバインディングの MIC を計算します。gss_accept_sec_context() は、MIC が一致しない場合に GSS_C_BAD_BINDINGS を返します。

gss_accept_sec_context() からは転送されたチャネルバインディングが返されるため、受け入れ側は、それらの値に基づいてセキュリティー検査を実行できます。たとえば *application_data* の値をセキュアデータベースに保存しておいたコードワードと比較したりできます。

注 – チャネルバインディング情報の機密性を提供するかどうかは、実際の機構によって異なります。したがって、アプリケーションは、機密性が保証されるまで、チャネルバインディングに機密情報を含めてはなりません。機密性が利用可能かどうかを判断するには、アプリケーション内で `gss_init_sec_context()` または `gss_accept_sec_context()` の *ret_flags* 引数を検査します。値 `GSS_C_CONF_FLAG` と `GSS_C_PROT_READY_FLAG` が機密性を示します。*ret_flags* については、78 ページの「GSS-API におけるコンテキストの起動」または 80 ページの「GSS-API におけるコンテキストの受け入れ」を参照してください。

機構はそれぞれ、チャネルバインディングにおけるアドレスとアドレス型に追加の制限を課すことができます。たとえば、機構は、チャネルバインディングの *initiator_address* フィールドが `gss_init_sec_context()` に返されるかどうかを検証したりできます。したがって、アプリケーションの移植性を高めるには、アドレスフィールドに正しい情報を設定する必要があります。正しい情報を決定できない場合は、`GSS_C_AF_NULLADDR` をアドレス型に指定する必要があります。

GSS-API におけるコンテキストのエクスポートとインポート

GSS-API は、コンテキストをエクスポートおよびインポートする方法を提供します。この機能を使えば、マルチプロセスアプリケーション (通常はコンテキスト受け入れ側) は、あるプロセスから別のプロセスにコンテキストを転送できます。たとえば、受け入れ側に、コンテキスト起動側からの応答を待つプロセスと、コンテキストに送信されたデータを使用するプロセスが存在する可能性があります。129 ページの「`test_import_export_context()` 関数の使用」では、これらの関数を使ってコンテキストを保存および復元する方法について説明しています。

関数 `gss_export_sec_context()` は、エクスポートされるコンテキストに関する情報が入ったプロセス間トークンを作成します。詳細は、74 ページの「GSS-API におけるプロセス間トークン」を参照してください。`gss_export_sec_context()` を呼び出す前に、トークンを受信するバッファーを `GSS_C_NO_BUFFER` に設定する必要があります。

次に、アプリケーションはそのトークンをほかのプロセスに渡します。新しいプロセスはそのトークンを受け入れ、それを `gss_import_sec_context()` に渡します。多くの場合、アプリケーション間でトークンを渡すときに使用される関数が、プロセス間でトークンを渡すときにも使用されます。

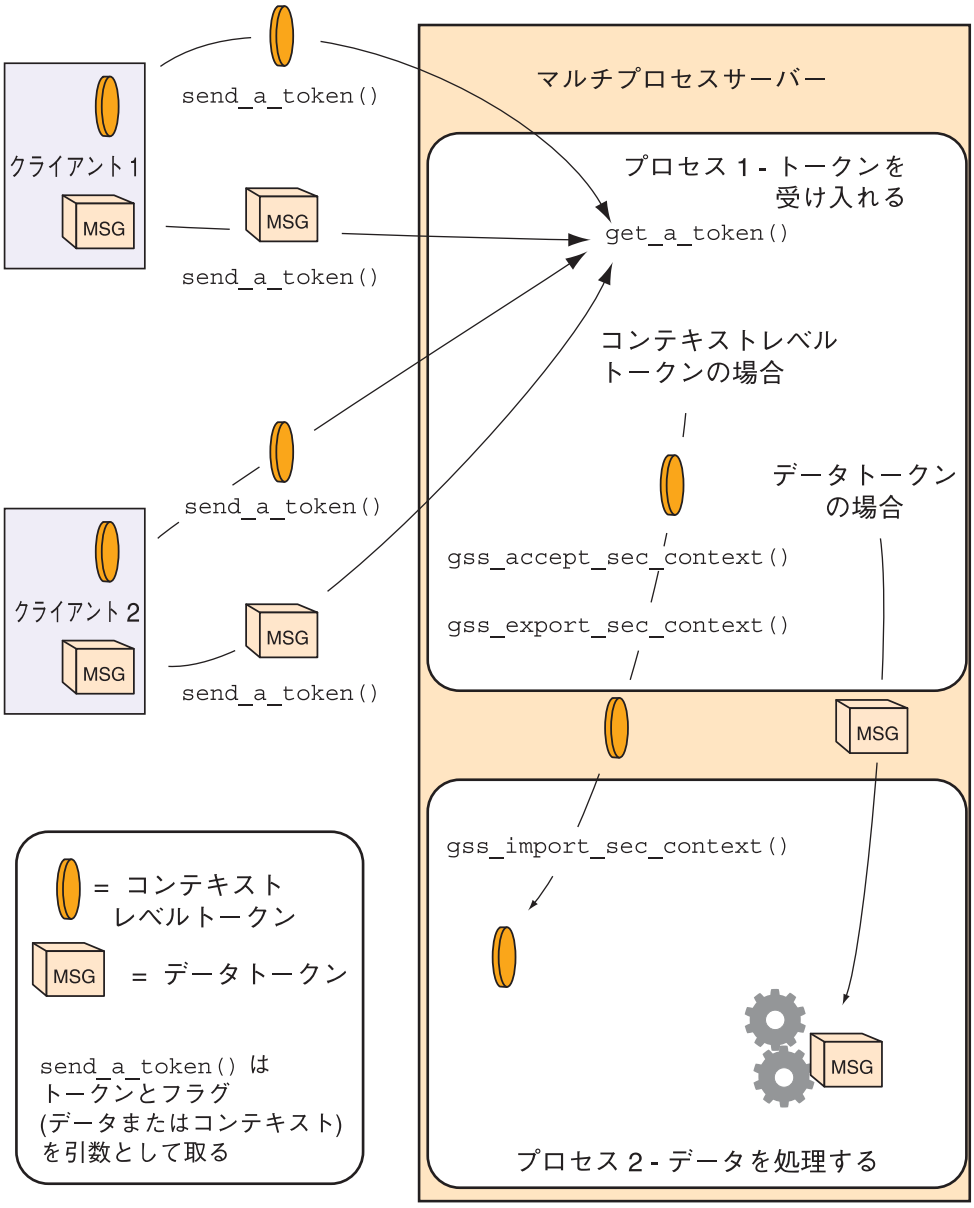
セキュリティープロセスのインスタンスは一度に 1 つしか存在できません。`gss_export_sec_context()` はエクスポートされたコンテキストを無効にし、そのコンテキストハンドルを `GSS_C_NO_CONTEXT` に設定します。また、`gss_export_sec_context()` は、そのコンテキストに関連付けられたプロセス内の

すべてのリソースも解放します。コンテキストのエクスポートを完了できない場合、gss_export_sec_context() は、既存のセキュリティーコンテキストを元のまま残し、プロセス間トークンも返しません。

すべての機構でコンテキストをエクスポートできるわけではありません。アプリケーションでは、gss_accept_sec_context() または gss_init_sec_context() の *ret_flags* 引数をチェックして、コンテキストをエクスポートできるかどうかを判定できます。このフラグに GSS_C_TRANS_FLAG が設定されている場合、コンテキストはエクスポートできます。80 ページの「GSS-API におけるコンテキストの受け入れ」と 78 ページの「GSS-API におけるコンテキストの起動」を参照してください。

図 4–6 に、マルチプロセスの受け入れ側がコンテキストをエクスポートしてマルチタスクを実現している様子を示します。この例では、プロセス 1 はトークンを受け取って処理します。このステップにより、コンテキストレベルトークンとデータトークンが分離され、それらのトークンがプロセス 2 に渡されます。プロセス 2 はアプリケーション固有の方法でデータを処理します。この図では、クライアントはすでに gss_init_sec_context() からエクスポートトークンを取得しています。クライアントはトークンをユーザー定義関数 send_a_token() に渡します。send_a_token は、転送するトークンがコンテキストレベルトークンまたはメッセージトークンのどちらであるかを示します。send_a_token() はトークンをサーバーに転送します。この図には示されていませんが、おそらく、send_a_token() はスレッド間でトークンを渡すときにも使用されます。

図 4–6 コンテキストのエクスポート:マルチスレッド化された受け入れ側の例



GSS-API におけるコンテキスト情報の取得

GSS-API は、指定されたセキュリティーコンテキストについての情報を取得する関数 gss_inquire_context(3GSS) を提供します。コンテキストは完全でなくてもかまわな

い点に注意してください。コンテキストハンドルを指定すると、gss_inquire_context() はそのコンテキストについて次の情報を提供します。

- コンテキスト起動側の名前
- コンテキスト受け入れ側の名前
- コンテキストが有効である時間 (秒)
- コンテキストで使用されるセキュリティー機構
- いくつかのコンテキストパラメータフラグ。これらのフラグは gss_accept_sec_context(3GSS) 関数の *ret_flags* 引数と同じです。これらのフラグは、委託や相互認証などを請け負います。80 ページの「GSS-API におけるコンテキストの受け入れ」を参照してください。
- 照会元のアプリケーションがコンテキスト起動側であるかどうかを示すフラグ
- コンテキストが完全に確立されているかどうかを示すフラグ

GSS-API における保護されたデータの送信

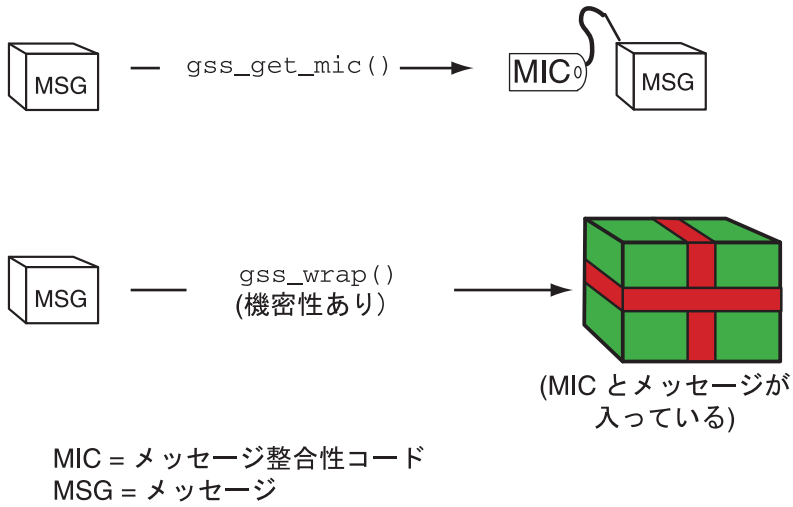
2 つのピア間でコンテキストが確立されたあと、メッセージを送信する前にそのメッセージを保護できます。

コンテキストの確立時に使用されるのは、もっとも基本的な GSS-API 保護である「認証」だけです。実際のセキュリティー機構によって異なりますが、GSS-API は次の 2 つの保護も提供します。

- 整合性 – gss_get_mic() 関数によってメッセージに対するメッセージ整合性コード (MIC) が生成されます。受信側は、MIC を検査することで、受信したメッセージが送信されたメッセージと同じかどうかを確認できます。
- 機密性 – MIC の使用に加え、メッセージが暗号化されます。暗号化を実行するのは、GSS-API の gss_wrap() 関数です。

gss_get_mic() と gss_wrap() の違いを、次の図に示します。gss_get_mic() を使用した場合、受信側はメッセージが変更されていないことを示すタグを受け取ります。gss_wrap() を使用した場合、受信側は暗号化されたメッセージとタグを受け取ります。

図 4–7 gss_get_mic() と gss_wrap()



使用すべき関数はユーザーの状況に応じて異なります。gss_wrap() は整合性サービスも含むため、多くのプログラムは gss_wrap() を使用します。プログラムは、機密性サービスが利用可能かどうかを判定できます。続いてプログラムは、その利用可能性に応じて、機密性を指定して、あるいは指定しないで、gss_wrap() を呼び出すことができます。使用例については、109 ページの「メッセージのラップと送信」を参照してください。ただし、gss_get_mic() を使用するとメッセージをラップ解除する必要がないため、gss_wrap() を使用する場合よりも CPU サイクルを節約できます。したがって、機密性が必要ないプログラムは、gss_get_mic() でメッセージを保護する可能性があります。

gss_get_mic() によるメッセージのタグ付け

gss_get_mic() を使用すると、プログラムは暗号化 MIC をメッセージに追加できます。受信側は、gss_verify_mic() でメッセージの MIC を検査できます。

gss_get_mic() は gss_wrap() とは対照的に、メッセージと MIC を別々に出力します。この分離は、送信側アプリケーションがメッセージと対応する MIC の両方を送信する必要があることを意味します。さらに重要なのは、受信側がメッセージと MIC を区別できる必要がある、という点です。メッセージと MIC を適切に処理するには、次のいずれかの方法を使用します。

- プログラム制御 (つまり、状態) を通じて。受信側アプリケーションは受信関数を 2 回呼び出す (つまり、1 回目はメッセージを取得するため、2 回目はメッセージの MIC を取得するため) ことをあらかじめ知ることができます。

- フラグを通じて。送信側と受信側は、どの種類のトークンを含めるかをフラグで示すことができます。
- メッセージと MIC の両方を含むユーザー定義トークン構造体を通じて。

gss_get_mic() が正常終了すると、GSS_S_COMPLETE が返されます。指定された QOP が有効でない場合、GSS_S_BAD_QOP が返されます。詳細は、gss_get_mic(3GSS) を参照してください。

gss_wrap() によるメッセージのラップ

メッセージは、gss_wrap() 関数でラップすることが可能です。gss_get_mic() と同様に、gss_wrap() は MIC を提供します。また、機密性が要求され、かつ実際の機構で利用可能である場合には、gss_wrap() は指定されたメッセージの暗号化も行います。メッセージの受信側は gss_unwrap() でメッセージをラップ解除します。

gss_wrap() は gss_get_mic() とは違い、メッセージと MIC をいっしょにラップし、1 つの出力メッセージにします。このバンドルを送信する関数の呼び出しは、1 回だけですみます。これに対し、gss_unwrap() はメッセージを抽出します。MIC はアプリケーションからは見えません。

メッセージが正常にラップされた場合、gss_wrap() は GSS_S_COMPLETE を戻します。要求された QOP が有効でない場合、GSS_S_BAD_QOP が返されます。gss_wrap() の使用例については、109 ページの「メッセージのラップと送信」を参照してください。

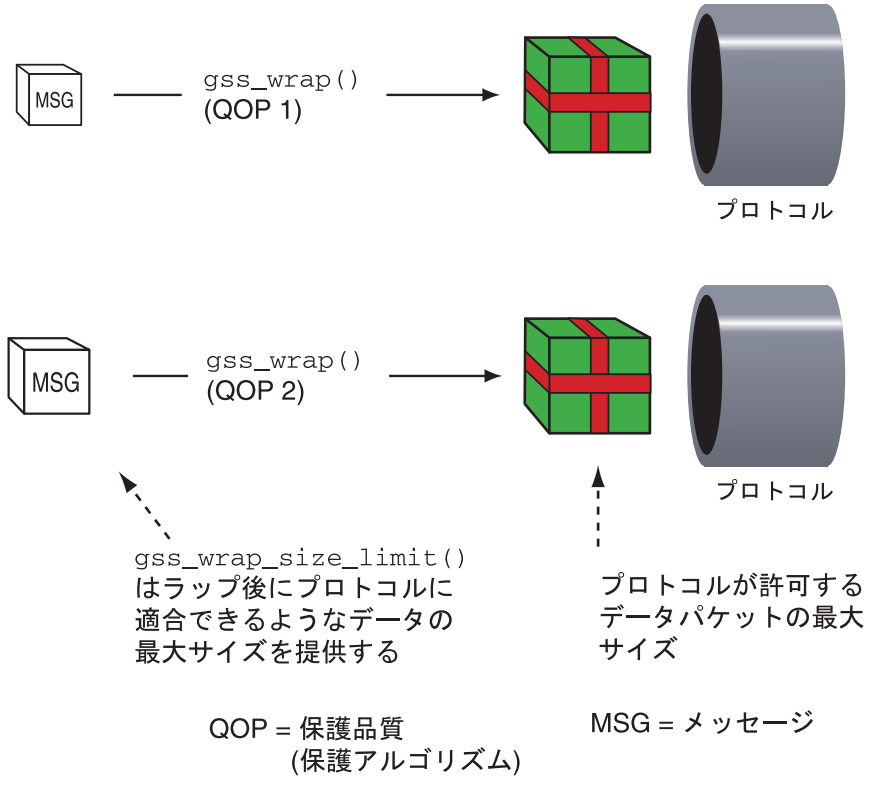
GSS-API におけるラップサイズ問題への対処法

gss_wrap() でメッセージをラップすると、送信すべきデータのサイズが増加します。保護されたメッセージパケットは、指定された転送プロトコルを通過するのに適したサイズである必要があります。したがって、GSS-API は関数 gss_wrap_size_limit() を提供しています。 gss_wrap_size_limit() は、プロトコルにとって大きすぎないサイズにラップ可能なメッセージの最大サイズを計算します。この最大サイズを超える場合、アプリケーションは gss_wrap() を呼び出す前にメッセージを分割できます。メッセージを実際にラップする前にラップサイズの制限値を必ず検査してください。

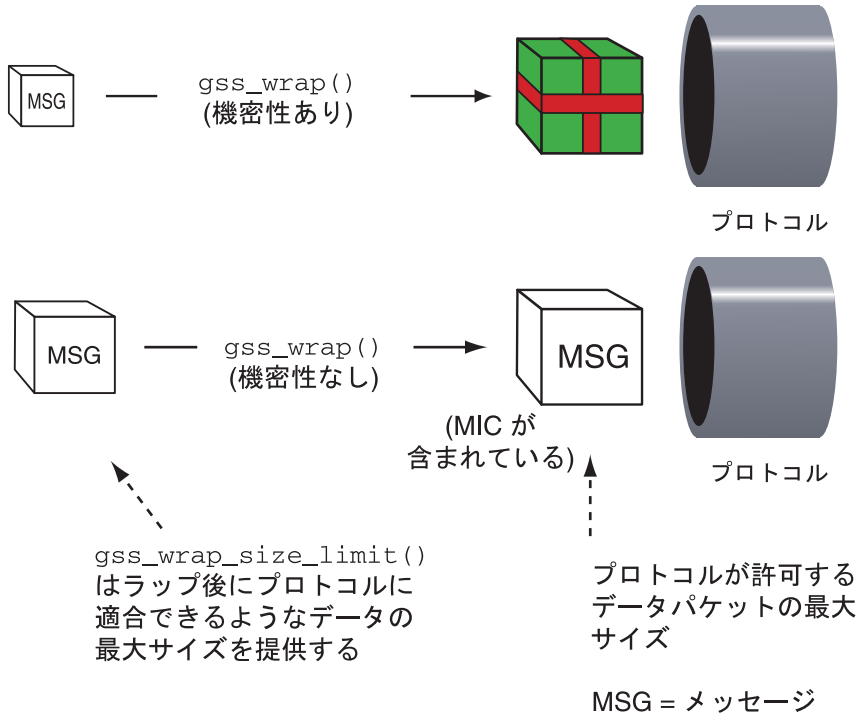
サイズの増加量は次の 2 つの要因に依存します。

- 変形を行うためにどの QOP アルゴリズムを使用するか
- 機密性を呼び出すかどうか

デフォルトの QOP は、GSS-API の実装ごとに異なる可能性があります。したがって、デフォルトの QOP を指定した場合でも、ラップ後のメッセージのサイズが異なる可能性があります。この可能性を示したのが次の図です。



機密性を適用するかどうかにかかわらず、gss_wrap() によってメッセージのサイズが増加します。gss_wrap() は、転送メッセージ内に MIC を埋め込みます。しかし、メッセージを暗号化すると (機密性を適用すると)、メッセージのサイズはさらに増加します。このプロセスを示したのが次の図です。



gss_wrap_size_limit() が正常終了すると、GSS_S_COMPLETE が返されます。指定された QOP が有効でない場合、GSS_S_BAD_QOP が返されます。gss_wrap_size_limit() で元のメッセージの最大サイズを求める例については、109 ページの「メッセージのラップと送信」を参照してください。

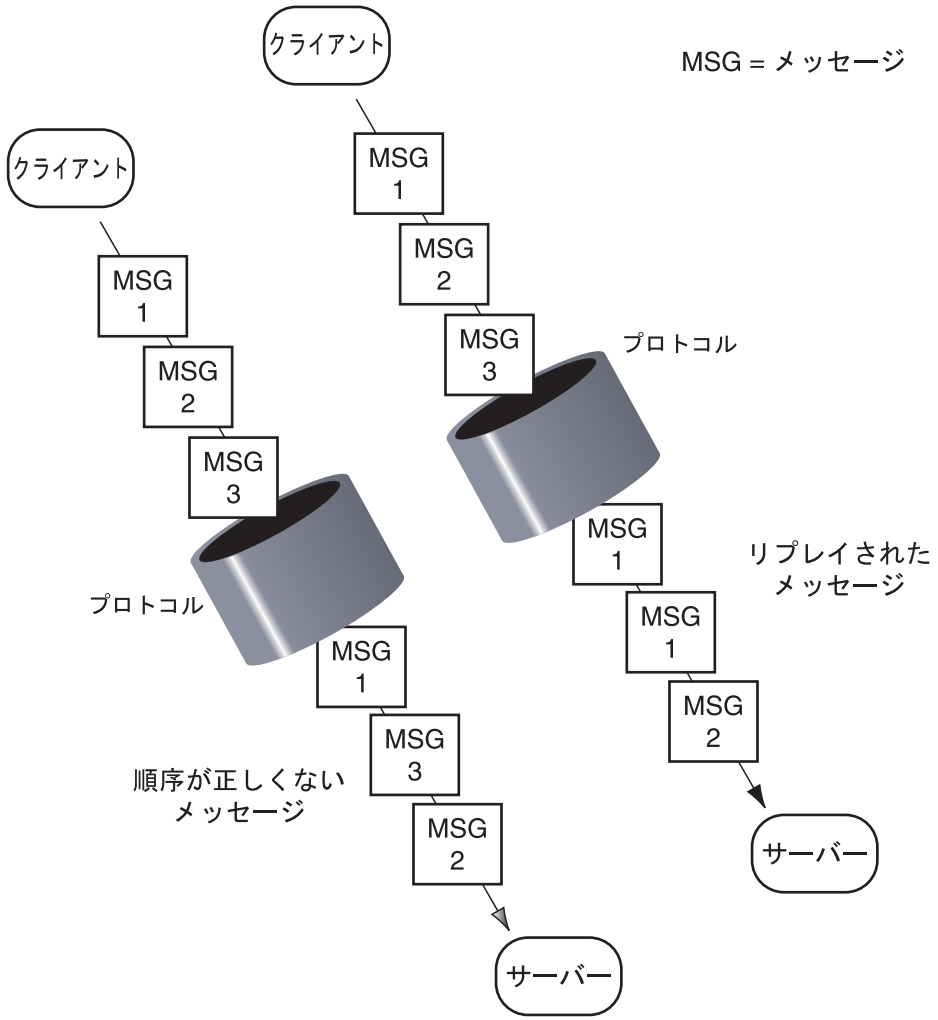
この呼び出しが正常に終了したとしても、gss_wrap() が *max-input-size* バイトの長さを持つメッセージを必ず保護できるという保証はありません。この機能は、gss_wrap() の呼び出し時点で必要なシステムリソースが利用可能かどうかに依存します。詳細は、gss_wrap_size_limit(3GSS) のマニュアルページを参照してください。

GSS-API における順序の問題の検出

コンテキスト起動側がコンテキスト受け入れ側に一連のデータパケットを順次転送する際、一部の機構では、その順序が正しいかどうかをコンテキスト受け入れ側が検査できるようになっています。これらの検査には、「パケットが正しい順序で到着したか」、「パケットが不必要に重複していないか」が含まれます。次の図を参

照してください。受け入れ側がこれら 2 つの状態を検査するのは、パケットの検証時とパケットのラップ解除時です。詳細は、128 ページの「メッセージのラップ解除」を参照してください。

図 4–8 リプレイされたメッセージと順序が正しくないメッセージ



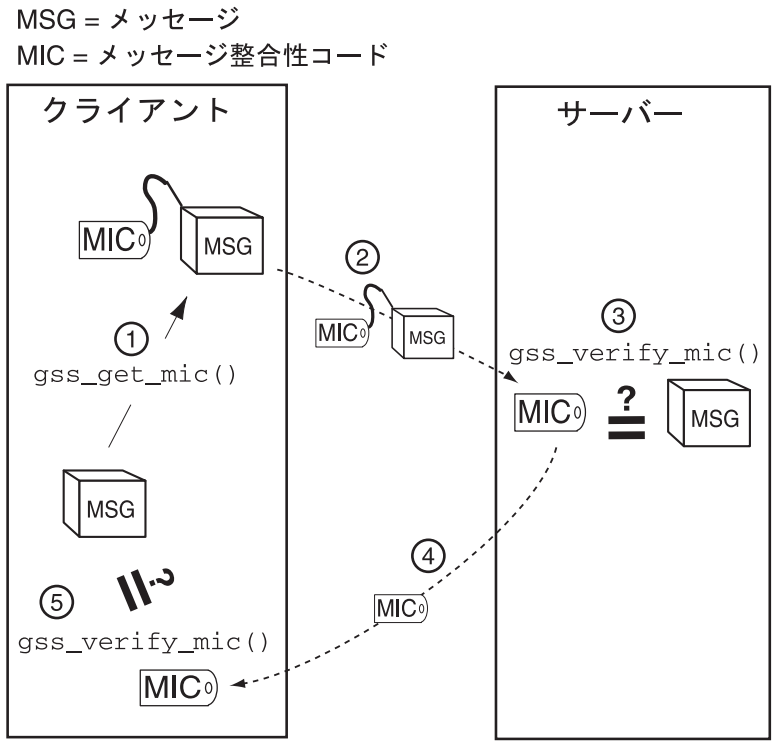
起動側は gss_init_sec_context() で順序を検査できます。それには、GSS_C_REPLAY_FLAG または GSS_C_SEQUENCE_FLAG を論理 OR で *req_flags* 引数に設定します。

GSS-API におけるメッセージ転送の確認

受信側は、転送メッセージのラップ解除後または検証後に、確認を送信側に返すことができます。つまり、そのメッセージの MIC を返送します。送信側がラップはしなかったが gss_get_mic() で MIC をタグ付けしているメッセージの場合を考えます。実行手順 (図 4–9) は次のようになります。

1. 起動側は gss_get_mic() でメッセージにタグ付けします。
2. 起動側はメッセージと MIC を受け入れ側に送信します。
3. 受け入れ側は gss_verify_mic() でメッセージを検証します。
4. 受け入れ側は MIC を起動側に返送します。
5. 起動側は gss_verify_mic() で、元のメッセージに対して受信した MIC を検証します。

図 4–9　MIC データの確認

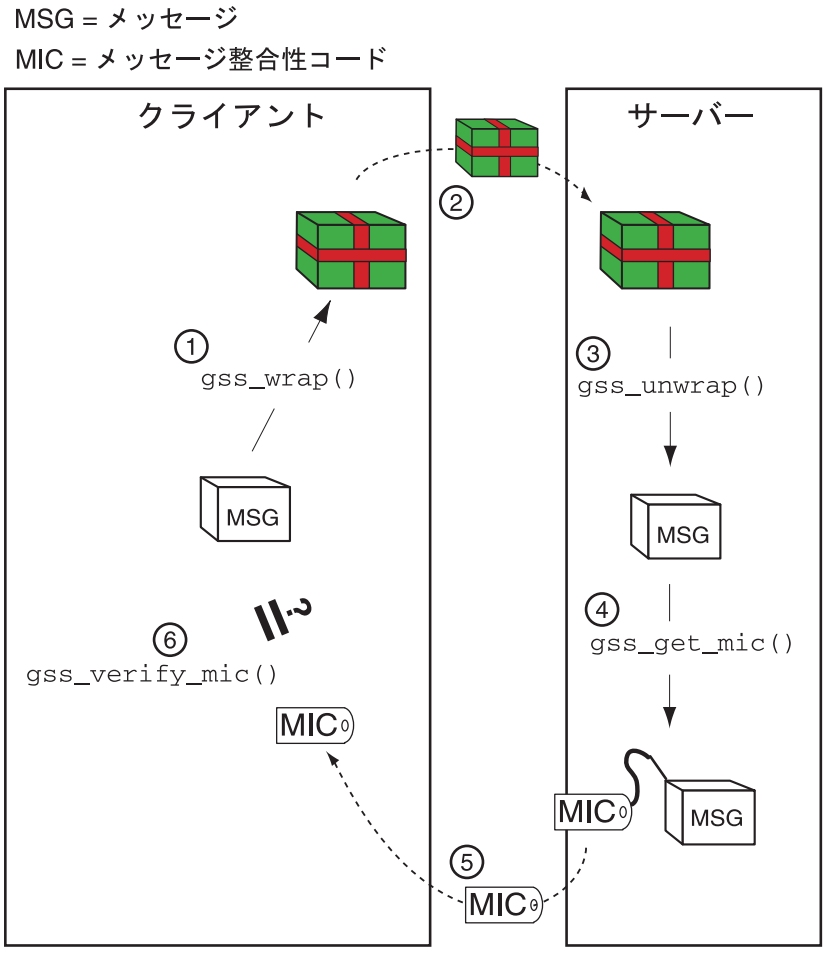


ラップされたデータの場合、`gss_unwrap()` 関数はメッセージと MIC を別々に生成しません。したがって、受信側は、受信した (およびラップ解除した) メッセージから MIC を生成する必要があります。実行手順 (図 4–10) は次のようになります。

1. 起動側は `gss_wrap()` でメッセージをラップします。
2. 起動側はラップしたメッセージを送信します。
3. 受け入れ側は `gss_unwrap()` でメッセージをラップ解除します。
4. 受け入れ側は `gss_get_mic()` でラップ解除されたメッセージの MIC を生成します。
5. 受け入れ側は抽出した MIC を起動側に返信します。
6. 起動側は `gss_verify_mic()` で、元のメッセージに対して受信した MIC を検証します。

アプリケーションは、GSS-API データ用に割り当てられたすべてのデータ領域を解放する必要があります。これに関係する関数
は、`gss_release_buffer(3GSS)`、`gss_release_cred(3GSS)`、`gss_release_name(3GSS)`、および `gss_release_oid_set(3GSS)` です。

図 4–10　ラップされたデータの確認



GSS-API セッションのクリーンアップ

最終的に、すべてのメッセージの送受信が完了し、起動側と受け入れ側のアプリケーションが終了します。この時点で、両アプリケーションは `gss_delete_sec_context()` を呼び出して共有コンテキストを破棄する必要があります。`gss_delete_sec_context()` はコンテキストに関連するローカルのデータ構造体を削除します。

用心のため、アプリケーションは GSS-API データ用に割り当てたデータ領域をすべて解放するべきです。このような関数には、`gss_release_buffer()`、`gss_release_cred()`、`gss_release_name()`、および `gss_release_oid_set()` があります。

◆ ◆ ◆ 第 5 章

GSS-API クライアント例

この章では、一般的な GSS-API クライアントアプリケーションについて段階的に説明します。次の項目について説明します。

GSS-API クライアント例の概要

クライアント側プログラム例 `gss-client` は、サーバーとのセキュリティーコンテキストを作成し、セキュリティーパラメータを確立し、文字列 (メッセージ) をサーバーに送信します。 このプログラムは接続時に、単純な TCP ベースのソケット接続を使用します。

以降では、`gss-client` がどのように動作するかを段階的に説明します。`gss-client` は、GSS-API の機能説明用に設計されたプログラム例であるため、関連部分についてのみ詳しく説明します。2 つのアプリケーションの完全なソースコードは付録に含まれています。また、次の場所からダウンロードすることもできます。

```
http://developers.sun.com/prodtech/solaris/downloads/index.html
```

GSS-API クライアント例の構造

`gss-client` アプリケーションは次の手順を実行します。

1. コマンド行を解析します。

2. 機構が指定されている場合、その機構のオブジェクト ID (OID) を作成します。それ以外の場合、デフォルトの機構が使用されます。これがごく普通の場合です。
3. サーバーとの接続を設定します。
4. セキュリティーコンテキストを確立します。
5. メッセージをラップして送信します。
6. サーバーが正しくメッセージに署名していることを検証します。
7. セキュリティーコンテキストを削除します。

GSS-API クライアント例の実行

次に、`gss-client` 例のコマンド行の書式を示します。

```
gss-client [-port port] [-d] [-mech mech] host service-name [-f] msg
```

- *port* – *host* で指定されたリモートマシンへの接続を確立するためのポート番号です。
- `-d` フラグ – サーバーへのセキュリティー資格の委託を可能にします。具体的には、*deleg-flag* 変数が GSS-API 値 `GSS_C_DELEG_FLAG` に設定されます。*d* フラグが設定されていない場合、deleg-flag は 0 に設定されます。
- *mech* – Kerberos v5 など、使用するセキュリティー機構の名前です。機構が指定されていない場合、GSS-API はデフォルトの機構を使用します。
- *host* – サーバーの名前です。
- *service-name* – クライアントが要求するネットワークサービスの名前です。そのようなサービスの一般的な例として、`telnet`、`ftp`、`login` などが挙げられます。
- *msg* – 保護されたデータとしてサーバーに送信される文字列です。`-f` オプションが指定されている場合、*msg* は文字列を読み取るべきファイル名です。

次に、クライアントアプリケーションプログラムの一般的なコマンド行の例を示します。

```
% gss-client -port 8080 -d -mech kerberos_v5 erebos.eng nfs "ls"
```

次の例では、機構、ポート、および委託が指定されていません。

```
% gss-client erebos.eng nfs "ls"
```

GSSAPI クライアント例: main() 関数

すべての C プログラムと同様に、プログラムの外部骨格はエントリポイント関数 main() に含まれます。main() は次の 4 つの機能を実行します。

- コマンド行引数を解析し、それらを変数に代入します。
- デフォルト以外の機構を使用する必要がある場合、parse_oid() を呼び出して GSS-API OID (オブジェクト識別子) を作成します。オブジェクト識別子はセキュリティー機構の名前から生成されます。ただし、それには機構名が指定される必要があります。
- call_server() を呼び出します。この関数は、コンテキストの作成とデータの送信を実際に行います。
- データの送信後、必要に応じて OID の記憶領域を解放します。

main() ルーチンのソースコードを、次に示します。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

例 5–1 gss-client 例: main()

```
int main(argc, argv)
     int argc;
     char **argv;
{
     char *msg;
     char service_name[128];
     char hostname[128];
     char *mechanism = 0;
     u_short port = 4444;
     int use_file = 0;
     OM_uint32 deleg_flag = 0, min_stat;

     display_file = stdout;

     /* Parse command-line arguments. */

        argc--; argv++;
     while (argc) {
          if (strcmp(*argv, "-port") == 0) {
               argc--; argv++;
               if (!argc) usage();
               port = atoi(*argv);
          } else if (strcmp(*argv, "-mech") == 0) {
               argc--; argv++;
               if (!argc) usage();
               mechanism = *argv;
          } else if (strcmp(*argv, "-d") == 0) {
               deleg_flag = GSS_C_DELEG_FLAG;
          } else if (strcmp(*argv, "-f") == 0) {
```

例 5–1 gss-client 例: main() (続き)

```
                use_file = 1;
           } else
                break;
           argc--; argv++;
      }
      if (argc != 3)
           usage();

      if (argc > 1) {
                strcpy(hostname, argv[0]);
         } else if (gethostname(hostname, sizeof(hostname)) == -1) {
                        perror("gethostname");
                        exit(1);
         }


      if (argc > 2) {
         strcpy(service_name, argv[1]);
         strcat(service_name, "@");
         strcat(service_name, hostname);

      }

       msg = argv[2];

      /* Create GSSAPI object ID. */
      if (mechanism)
          parse_oid(mechanism, &g_mechOid);

      /* Call server to create context and send data. */
      if (call_server(hostname, port, g_mechOid, service_name,
                   deleg_flag, msg, use_file) < 0)
           exit(1);

      /* Release storage space for OID, if still allocated  */
      if (g_mechOid != GSS_C_NULL_OID)
          (void) gss_release_oid(&min_stat, &gmechOid);

      return 0;
}
```

サーバーとの接続のオープン

call_server() 関数は、次のコードを使ってサーバーとの接続を確立します。

```
      if ((s = connect_to_server(host, port)) < 0)
           return -1;
```

s は int 型のファイル記述子であり、最初は socket() の呼び出しから戻されます。

connect_to_server() はソケット経由で接続を作成する単純な関数であり、GSS-API を使用していません。connect_to_server() のソースコードを、次に示します。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

例 5–2 connect_to_server() 関数

```
int connect_to_server(host, port)
     char *host;
     u_short port;
{
     struct sockaddr_in saddr;
     struct hostent *hp;
     int s;

     if ((hp = gethostbyname(host)) == NULL) {
          fprintf(stderr, "Unknown host: %s\n", host);
          return -1;
     }

     saddr.sin_family = hp->h_addrtype;
     memcpy((char *)&saddr.sin_addr, hp->h_addr, sizeof(saddr.sin_addr));
     saddr.sin_port = htons(port);

     if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
          perror("creating socket");
          return -1;
     }
     if (connect(s, (struct sockaddr *)&saddr, sizeof(saddr)) < 0) {
          perror("connecting to server");
          (void) close(s);
          return -1;
     }

     return s;
}
```

サーバーとのセキュリティーコンテキストの確立

接続が確立されたあと、call_server() は、次のように関数 client_establish_context() を使ってセキュリティーコンテキストを作成します。

```
if (client_establish_context(s, service-name, deleg-flag, oid, &context,
                                  &ret-flags) < 0) {
          (void) close(s);
          return -1;
     }
```

- *s* は、connect_to_server() で確立された接続を表すファイル記述子です。
- *service-name* は、要求されたネットワークサービスです。
- *deleg-flag* は、サーバーがクライアントのプロキシとして動作できるかどうかを指定します。

- *oid* は機構です。
- *context* は作成されるコンテキストです。
- *ret-flags* は、GSS-API 関数 `gss_init_sec_context()` から戻される任意のフラグを表す `int` です。

`client_establish_context()` は次の作業を実行します。

- サービス名を GSS-API 内部形式に変換します
- セキュリティーコンテキストが完了するまで、クライアントとサーバー間のトークン交換ループを実行します

サービス名の GSS-API 形式への変換

`client_establish_context()` が実行する最初の作業は、`gss_import_name()` を使ってサービス名の文字列を GSS-API 内部形式に変換することです。

例 5–3 `client_establish_context()` – サービス名の変換

```
     /*
     * Import the name into target_name.  Use send_tok to save
     * local variable space.
     */

     send_tok.value = service_name;
     send_tok.length = strlen(service_name) + 1;
     maj_stat = gss_import_name(&min_stat, &send_tok,
                        (gss_OID) GSS_C_NT_HOSTBASED_SERVICE, &target_name);
     if (maj_stat != GSS_S_COMPLETE) {
          display_status("parsing name", maj_stat, min_stat);
          return -1;
     }
```

`gss_import_name()` は、サービスの名前を不透明な GSS_API バッファー `send_tok` として受け取り、その文字列を GSS_API 内部名 `target_name` に変換します。 `send_tok` は新しい `gss_buffer_desc` を宣言せず、領域を節約するために使用されます。3 番目の引数は `gss_OID` 型で、`send_tok` に格納されている名前の形式を示します。この例は `GSS_C_NT_HOSTBASED_SERVICE` で、サービスの形式が *service@host* であることを意味します。この引数に指定可能なその他の値については、235 ページの「名前型」を参照してください。

GSS-API セキュリティーコンテキストの確立

サービスの GSS-API 内部形式への変換が完了すると、コンテキストを確立できます。移植性を最大にするには、コンテキストの確立を常にループとして実行する必要があります。

ループに入る前に、client_establish_context() は、コンテキストと *token_ptr* パラメータを初期化します。*token_ptr* の使用には選択肢があります。*token_ptr* がポイントできるのは、サーバーに送信するトークンである *send_tok*、サーバーから返送されてきたトークンである *recv_tok* のいずれかです。

ループの内側では、次の 2 つの項目が検査されます。

- gss_init_sec_context() から戻される状態

  戻り状態は、ループの異常終了を要求する可能性のあるすべてのエラーを捕捉します。gss_init_sec_context() が GSS_S_CONTINUE_NEEDED を戻すのは、別の送信すべきトークンがサーバー側に存在する場合に限ります。
- サーバーに送信すべきトークンのサイズ (gss_init_sec_context() によって生成される)

  トークンサイズ 0 は、サーバーに送信できる情報がこれ以上存在しないことと、ループが終了可能であることを意味します。トークンサイズは *token_ptr* で決定されます。

次に、このループの擬似コードを示します。

```
do
   gss_init_sec_context()
   if (コンテキストが作成されなかった場合)
      エラーを出力して終了する
  if (状態が「完了」または「処理中」のどちらでもない場合)
      サービスの名前空間を解放し、エラーを出力して終了する
  if (サーバーに送信するトークンがある場合、つまりサイズが 0 以外の場合)
      トークンを送信する
     if (トークンの送信が失敗した場合)
         トークンとサービスの名前空間を解放し、エラーを出力して終了する
     送信し終わったトークンの名前空間を解放する
   if (コンテキストの確立が完了していない場合)
      サーバーからトークンを受信する
while (コンテキストが完了していない)
```

ループの最初で、gss_init_sec_context() が呼び出されます。この関数の引数は次のとおりです。

- 実際の機構が設定する状態コード。
- 資格ハンドル。例ではデフォルトの主体として動作させるために、GSS_C_NO_CREDENTIAL を使用します。
- 作成するコンテキストハンドル。
- コンテキスト受け入れ側の名前。
- 必要な機構のオブジェクト ID。

- 要求フラグ。この場合にクライアントが要求することは、サーバーが自分自身を認証すること、メッセージの複製をオンにすること、要求された場合にサーバーがプロキシとして動作すること、のいずれかです。
- コンテキストの時間制限はありません。
- チャネルバインディングの要求はありません。
- ピアアプリケーションから受信するトークン。
- サーバーが実際に使用する機構。アプリケーションがこの値を使用しないため、ここでは NULL に設定されています。
- ピアアプリケーションに送信するトークン。これは、gss_init_sec_context() が作成するトークンです。
- 戻りフラグ。この例では無視するため、NULL に設定されています。

注 – クライアントは、コンテキストの起動前に資格を取得する必要はありません。クライアント側では、資格の管理は GSS-API によって透過的に処理されます。つまり、この主体のためにこの機構が作成した資格をどのように取得するかを、GSS-API は知っているということです。このため、アプリケーションは gss_init_sec_context() にデフォルトの資格を渡しています。しかし、サーバー側では、サーバーアプリケーションはコンテキストを受け入れる前に、サービスの資格を明示的に獲得する必要があります。118 ページの「資格の獲得」を参照してください。

---

connect_to_server() は、コンテキストまたはその一部が存在しており、かつ gss_init_sec_context() が有効な状態を戻していることを確認したあと、gss_init_sec_context() がサーバーに送信すべきトークンを提供しているかどうかを検査します。トークンが存在しない場合、それはトークンがこれ以上必要ないことを、サーバーが示していると考えられます。トークンが提供された場合、そのトークンをサーバーに送信する必要があります。トークンの送信に失敗した場合、トークンとサービスの名前空間を決定できないため、connect_to_server() が終了します。次のアルゴリズムは、トークンの長さを調べることでトークンの存在の有無を検査しています。

```
if (send_tok_length != 0) {
     if (send_token(s, &send_tok) < 0) {
          (void) gss_release_buffer(&min_stat, &send_tok);
          (void) gss_release_name(&min_stat, &target_name);
          return -1;
     }
}
```

send_token() は GSS-API 関数ではなく、ユーザーによって記述される必要があります。send_token() 関数は、トークンをファイル記述子に書き込みます。send_token()

は、正常終了時に 0 を、エラー時に –1 を戻します。GSS-API 自身はトークンの送受信を行いません。GSS-API によって作成されたトークンを送受信することは、呼び出し元のアプリケーションの責任です。

コンテキスト確立ループのソースコードを、次に示します。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

例 5–4 コンテキスト確立用のループ

```
/*
 * Perform the context establishment loop.
 *
 * On each pass through the loop, token_ptr points to the token
 * to send to the server (or GSS_C_NO_BUFFER on the first pass).
 * Every generated token is stored in send_tok which is then
 * transmitted to the server; every received token is stored in
 * recv_tok, which token_ptr is then set to, to be processed by
 * the next call to gss_init_sec_context.
 *
 * GSS-API guarantees that send_tok's length will be non-zero
 * if and only if the server is expecting another token from us,
 * and that gss_init_sec_context returns GSS_S_CONTINUE_NEEDED if
 * and only if the server has another token to send us.
 */

token_ptr = GSS_C_NO_BUFFER;
*gss_context = GSS_C_NO_CONTEXT;
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456

do {
    maj_stat =
        gss_init_sec_context(&min_stat, GSS_C_NO_CREDENTIAL,
        gss_context, target_name, oid,
        GSS_C_MUTUAL_FLAG | GSS_C_REPLAY_FLAG | deleg_flag,
        0, NULL,                        /* no channel bindings */
        token_ptr, NULL,                /* ignore mech type */
        &send_tok, ret_flags, NULL);  /* ignore time_rec */
    if (gss_context == NULL){
        printf("Cannot create context\n");
        return GSS_S_NO_CONTEXT;
    }
    if (token_ptr != GSS_C_NO_BUFFER)
        (void) gss_release_buffer(&min_stat, &recv_tok);
    if (maj_stat!=GSS_S_COMPLETE && maj_stat!=GSS_S_CONTINUE_NEEDED) {
        display_status("initializing context", maj_stat, min_stat);
        (void) gss_release_name(&min_stat, &target_name);
        return -1;
    }

    if (send_tok.length != 0){
        fprintf(stdout, "Sending init_sec_context token (size=%ld)...",
            send_tok.length);
```

例 5–4 コンテキスト確立用のループ (続き)

```
        if (send_token(s, &send_tok) < 0) {
            (void) gss_release_buffer(&min_stat, &send_tok);
            (void) gss_release_name(&min_stat, &target_name);
            return -1;
        }
    }
    (void) gss_release_buffer(&min_stat, &send_tok);

    if (maj_stat == GSS_S_CONTINUE_NEEDED) {
        fprintf(stdout, "continue needed...");
        if (recv_token(s, &recv_tok) < 0) {
            (void) gss_release_name(&min_stat, &target_name);
            return -1;
        }
        token_ptr = &recv_tok;
    }
    printf("\n");
} while (maj_stat == GSS_S_CONTINUE_NEEDED);
```

`send_token()` と `recv_token()` の動作方法についての詳細は、219 ページの「その他の GSS-API 関数例」を参照してください。

クライアント側におけるその他の GSS-API コンテキスト操作

プログラム例として、`gss-client` はいくつかの関数をデモ目的で実行しています。次のソースコードは基本作業に不可欠なものではありませんが、次の操作の使用法を示す目的で掲載します。

- コンテキストの保存と復元
- コンテキストフラグの表示
- コンテキストの状態の取得

これらの操作を含むソースコードを、次に示します。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。`http://www.sun.com/download/products.xml?id=41912db5` を参照してください。

例 5–5 gss-client: `call_server()` コンテキストの確立

```
/* Save and then restore the context */
     maj_stat = gss_export_sec_context(&min_stat,
                                       &context,
                                       &context_token);
     if (maj_stat != GSS_S_COMPLETE) {
```

例 5–5 gss-client: call_server() コンテキストの確立 *(続き)*

```
            display_status("exporting context", maj_stat, min_stat);
            return -1;
    }
    maj_stat = gss_import_sec_context(&min_stat,
                                      &context_token,
                                      &context);
    if (maj_stat != GSS_S_COMPLETE) {
       display_status("importing context", maj_stat, min_stat);
       return -1;
    }
    (void) gss_release_buffer(&min_stat, &context_token);

    /* display the flags */
    display_ctx_flags(ret_flags);

    /* Get context information */
    maj_stat = gss_inquire_context(&min_stat, context,
                                   &src_name, &targ_name, &lifetime,
                                   &mechanism, &context_flags,
                                   &is_local,
                                   &is_open);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("inquiring context", maj_stat, min_stat);
        return -1;
    }

    if (maj_stat == GSS_S_CONTEXT_EXPIRED) {
    printf(" context expired\n");
        display_status("Context is expired", maj_stat, min_stat);
        return -1;
    }
```

メッセージのラップと送信

gss-client アプリケーションは、データを送信する前にデータをラップ (つまり暗号化) する必要があります。アプリケーションは、次の手順に従ってメッセージをラップします。

- ラップサイズの制限値を決定します。この処理により、ラップ後のメッセージをプロトコルが確実に処理できることが保証されます。
- ソース名とターゲット名を取得します。名前をオブジェクト識別子から文字列に変換します。
- 機構名の一覧を取得します。名前をオブジェクト識別子から文字列に変換します。
- メッセージをバッファーに挿入し、メッセージをラップします。
- サーバーにメッセージを送信します。

メッセージをラップするソースコードを、次に示します。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

例 5–6 gss-client 例: call_server() – メッセージのラップ

```
/* Test gss_wrap_size_limit */
maj_stat = gss_wrap_size_limit(&min_stat, context, conf_req_flag,
    GSS_C_QOP_DEFAULT, req_output_size, &max_input_size);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("wrap_size_limit call", maj_stat, min_stat);
} else
    fprintf (stderr, "gss_wrap_size_limit returned "
        "max input size = %d \n"
        "for req_output_size = %d with Integrity only\n",
         max_input_size , req_output_size , conf_req_flag);

conf_req_flag = 1;
maj_stat = gss_wrap_size_limit(&min_stat, context, conf_req_flag,
    GSS_C_QOP_DEFAULT, req_output_size, &max_input_size);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("wrap_size_limit call", maj_stat, min_stat);
} else
    fprintf (stderr, "gss_wrap_size_limit returned "
        " max input size = %d \n" "for req_output_size = %d with "
        "Integrity & Privacy \n", max_input_size , req_output_size );

maj_stat = gss_display_name(&min_stat, src_name, &sname, &name_type);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("displaying source name", maj_stat, min_stat);
    return -1;
}

maj_stat = gss_display_name(&min_stat, targ_name, &tname,
    (gss_OID *) NULL);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("displaying target name", maj_stat, min_stat);
    return -1;
 }
fprintf(stderr, "\"%.*s\" to \"%.*s\", lifetime %u, flags %x, %s, %s\n",
    (int) sname.length, (char *) sname.value, (int) tname.length,
    (char *) tname.value, lifetime, context_flags,
    (is_local) ? "locally initiated" : "remotely initiated",
    (is_open) ? "open" : "closed");

(void) gss_release_name(&min_stat, &src_name);
(void) gss_release_name(&min_stat, &targ_name);
(void) gss_release_buffer(&min_stat, &sname);
(void) gss_release_buffer(&min_stat, &tname);

maj_stat = gss_oid_to_str(&min_stat, name_type, &oid_name);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("converting oid->string", maj_stat, min_stat);
    return -1;
}
fprintf(stderr, "Name type of source name is %.*s.\n", (int) oid_name.length,
```

例 5–6 gss-client 例: call_server() – メッセージのラップ (続き)

```
    (char *) oid_name.value);
(void) gss_release_buffer(&min_stat, &oid_name);

/* Now get the names supported by the mechanism */
maj_stat = gss_inquire_names_for_mech(&min_stat, mechanism, &mech_names);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("inquiring mech names", maj_stat, min_stat);
    return -1;
}

maj_stat = gss_oid_to_str(&min_stat, mechanism, &oid_name);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("converting oid->string", maj_stat, min_stat);
    return -1;
}
mechStr = (char *)__gss_oid_to_mech(mechanism);
fprintf(stderr, "Mechanism %.*s (%s) supports %d names\n", (int) oid_name.length,
    (char *) oid_name.value, (mechStr == NULL ? "NULL" : mechStr),
    mech_names->count);
(void) gss_release_buffer(&min_stat, &oid_name);

for (i=0; i < mech_names->count; i++) {
    maj_stat = gss_oid_to_str(&min_stat, &mech_names->elements[i], &oid_name);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("converting oid->string", maj_stat, min_stat);
        return -1;
    }
    fprintf(stderr, "  %d: %.*s\n", i, (int) oid_name.length, (
    char *) oid_name.value);

    (void) gss_release_buffer(&min_stat, &oid_name);
    }
(void) gss_release_oid_set(&min_stat, &mech_names);

if (use_file) {
    read_file(msg, &in_buf);
} else {
    /* Wrap the message */
    in_buf.value = msg;
    in_buf.length = strlen(msg) + 1;
}

if (ret_flag & GSS_C_CONF_FLAG) {
    state = 1;
else
    state = 0;
}
maj_stat = gss_wrap(&min_stat, context, 1, GSS_C_QOP_DEFAULT, &in_buf,
    &state, &out_buf);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("wrapping message", maj_stat, min_stat);
    (void) close(s);
    (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
    return -1;
} else if (! state) {
    fprintf(stderr, "Warning!  Message not encrypted.\n");
```

例 5–6 gss-client 例: call_server() – メッセージのラップ (続き)

```
}

/* Send to server */
if (send_token(s, &out_buf) < 0) {
    (void) close(s);
    (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
    return -1;
}
(void) gss_release_buffer(&min_stat, &out_buf);
```

GSS-API クライアントにおける署名ブロックの読み取りと検証

gss-client プログラムはこの段階で、送信したメッセージの有効性を検証できます。サーバーは、送信メッセージに対する MIC を戻します。そのメッセージは recv_token() を使って取得できます。

そして、gss_verify_mic() 関数を使ってメッセージの「署名」つまり MIC を検証します。gss_verify_mic() は、受け取った MIC を元のラップされていないメッセージと比較します。受け取った MIC は、*out_buf* に格納されたサーバーのトークンから取得します。ラップされていないメッセージの MIC は、*in_buf* 内に格納されています。2 つの MIC が一致した場合、メッセージの有効性は検証されたことになります。その後、クライアントは受け取ったトークンのバッファー (*out_buf*) を解放します。

次のソースコードは、署名ブロックの読み取りと検証を行う方法を示しています。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

例 5–7 gss-client 例 – 署名ブロックの読み取りと検証

```
/* Read signature block into out_buf */
     if (recv_token(s, &out_buf) < 0) {
          (void) close(s);
          (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
          return -1;
     }

/* Verify signature block */
     maj_stat = gss_(&min_stat, context, &in_buf,
                                &out_buf, &qop_state);
     if (maj_stat != GSS_S_COMPLETE) {
          display_status("verifying signature", maj_stat, min_stat);
          (void) close(s);
```

例 5–7 gss-client 例 – 署名ブロックの読み取りと検証 (続き)

```
          (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
          return -1;
     }
     (void) gss_release_buffer(&min_stat, &out_buf);

     if (use_file)
         free(in_buf.value);

     printf("Signature verified.\n");
```

セキュリティーコンテキストの削除

call_server() 関数は最後に、コンテキストを削除したあと、main() 関数に戻ります。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

例 5–8 gss-client 例: call_server() – コンテキストの削除

```
/* Delete context */
     maj_stat = gss_delete_sec_context(&min_stat, &context, &out_buf);
     if (maj_stat != GSS_S_COMPLETE) {
          display_status("deleting context", maj_stat, min_stat);
          (void) close(s);
          (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
          return -1;
     }

     (void) gss_release_buffer(&min_stat, &out_buf);
     (void) close(s);
     return 0;
```

◆ ◆ ◆ 第 6 章

GSS-API サーバー例

この章では、gss-server サンプルプログラムのソースコードについて段階的に説明します。次の項目について説明します。

GSS-API サーバー例の概要

サーバー側プログラム例 gss-server は、前章で説明した gss-client と連携して動作します。gss-server の基本目的は、gssapi-client からラップ済みメッセージを受け取り、そのメッセージに署名して戻すことです。

以降では、gss-server がどのように動作するかを段階的に説明します。gss-server は GSS-API の機能説明用のプログラム例であるため、関連する部分だけを詳しく説明します。2 つのアプリケーションの完全なソースコードは付録に含まれています。また、次の場所からダウンロードすることもできます。

```
http://developers.sun.com/prodtech/solaris/downloads/index.html
```

GSS-API サーバー例の構造

gss-structure アプリケーションは次の手順を実行します。

1. コマンド行を解析します。
2. 機構が指定された場合、その機構名を内部形式に変換します。
3. 呼び出し側の資格を獲得します。

4. inetd デーモンを使って接続するようにユーザーが指定しているかどうかを検査します。
5. クライアントとの接続を確立します。
6. クライアントからのデータを受信します。
7. データに署名して戻します。
8. 名前空間を解放し、終了します。

GSS-API サーバー例の実行

次に、gss-server のコマンド行の書式を示します。

```
gss-server [-port port] [-verbose] [-inetd] [-once] [-logfile file] \
                [-mech mechanism] service-name
```

- *port* は、応答を待つポートの番号です。port が指定されていない場合、プログラムはデフォルトでポート 4444 を使用します。
- -verbose を指定すると、gss-server 実行時にメッセージが表示されます。
- -inetd は、プログラムが inetd デーモンを使ってポートで応答を待つべきであることを示します。-inetd は、stdin と stdout を使ってクライアントに接続します。
- -once を指定すると、1 つの接続しか作成されなくなります。
- *mechanism* は、使用するセキュリティー機構 (Kerberos v5 など) の名前です。機構が指定されていない場合、GSS-API はデフォルトの機構を使用します。
- *service-name* は、クライアントが要求するネットワークサービス (telnet、ftp、login など) の名前です。

次に、一般的なコマンド行の例を示します。

```
% gss-server -port 8080 -once -mech kerberos_v5 erebos.eng nfs "hello"
```

GSSAPI サーバー例: main() 関数

gss-server の main() 関数は次の作業を実行します。

- コマンド行引数を解析し、それらを変数に代入します
- 機構に対応するサービスの資格を獲得します
- sign_server() 関数を呼び出します。この関数は、メッセージに署名して戻す処理にかかわる作業を実行します
- 獲得した資格を解放します
- 機構 OID の名前空間を解放します

- 接続を閉じます (まだ開いている場合)

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

例 6–1 gss-server 例: main()

```
int
main(argc, argv)
     int argc;
     char **argv;
{
     char *service_name;
     gss_cred_id_t server_creds;
     OM_uint32 min_stat;
     u_short port = 4444;
     int s;
     int once = 0;
     int do_inetd = 0;

     log = stdout;
     display_file = stdout;

     /* Parse command-line arguments. */
     argc--; argv++;
     while (argc) {
     if (strcmp(*argv, "-port") == 0) {
          argc--; argv++;
          if (!argc) usage();
          port = atoi(*argv);
     } else if (strcmp(*argv, "-verbose") == 0) {
          verbose = 1;
     } else if (strcmp(*argv, "-once") == 0) {
          once = 1;
     } else if (strcmp(*argv, "-inetd") == 0) {
          do_inetd = 1;
     } else if (strcmp(*argv, "-logfile") == 0) {
          argc--; argv++;
          if (!argc) usage();
          log = fopen(*argv, "a");
          display_file = log;
          if (!log) {
          perror(*argv);
          exit(1);
          }
     } else
          break;
     argc--; argv++;
     }
     if (argc != 1)
          usage();

     if ((*argv)[0] == '-')
          usage();
```

例 6–1 gss-server 例: main() (続き)

```
     service_name = *argv;

     /* Acquire service credentials. */
     if (server_acquire_creds(service_name, &server_creds) < 0)
          return -1;

     if (do_inetd) {
          close(1);
          close(2);
          /* Sign and return message. */
          sign_server(0, server_creds);
          close(0);
     } else {
          int stmp;

          if ((stmp = create_socket(port)) >= 0) {
              do {
                  /* Accept a TCP connection */
                  if ((s = accept(stmp, NULL, 0)) < 0) {
                      perror("accepting connection");
                      continue;
                  }
                  /* This return value is not checked, because there is
                     not really anything to do if it fails. */
                  sign_server(s, server_creds);
                  close(s);
              } while (!once);

              close(stmp);
          }
     }

     /* Close down and clean up. */
     (void) gss_release_cred(&min_stat, &server_creds);

     /*NOTREACHED*/
     (void) close(s);
     return 0;
}
```

資格の獲得

資格は、クライアントアプリケーション、サーバーアプリケーション、または GSS-API によって作成されるのではなく、基盤となる機構によって作成されます。クライアントプログラムは通常、ログイン時に取得された資格を持ちます。サーバーは常に、資格を明示的に獲得する必要があります。

gss-server プログラムは、提供するサービスの資格を取得するための関数 server_acquire_creds() を持っています。server_acquire_creds() は、入力としてサービス名と使用するセキュリティー機構を受け取ります。server_acquire_creds() は、その後、サービスの資格を戻します。

server_acquire_creds() は GSS-API 関数 gss_acquire_cred() を使って、サーバーが提供するサービスの資格を取得します。server_acquire_creds() が gss_acquire_cred() にアクセスする前に、server_acquire_creds() は次の 2 つの作業を行う必要があります。

1. 資格を取得できるように、機構リストの中身を検査し、そのリストに単一の機構だけが含まれるようにします。

   1 つの資格を複数の機構で共有できる場合、gss_acquire_cred() 関数はこのような機構すべての資格を戻します。したがって、gss_acquire_cred() は入力として機構の「セット」を受け取ります。76 ページの「GSS-API における資格の操作」を参照してください。しかしながら、この例も含めてほとんどの場合、単一の資格が複数の機構で動作することはありません。gss-server プログラムでは、コマンド行から単一の機構が指定されるか、またはデフォルトの機構が使用されます (指定されなかった場合)。したがって、最初の作業は、gss_acquire_cred() に渡される機構セットに 1 つの機構 (デフォルトまたはそれ以外) だけが入っていることを確認することです。次にコードを示します。

```
if (mechOid != GSS_C_NULL_OID) {
     desiredMechs = &mechOidSet;
     mechOidSet.count = 1;
     mechOidSet.elements = mechOid;
} else
     desiredMechs = GSS_C_NULL_OID_SET;
```

   GSS_C_NULL_OID_SET は、デフォルトの機構を使用することを示します。

2. サービス名を GSS-API 形式に変換します。

   gss_acquire_cred() が受け取るサービス名は gss_name_t 構造体の形式であるため、サービス名をその形式にインポートする必要があります。gss_import_name() 関数がこの変換作業を行います。ほかの GSS-API 関数と同様に、この関数でも引数は GSS-API 型である必要があるため、まず、サービス名を GSS-API バッファーにコピーする必要があります。次にコードを示します。

```
     name_buf.value = service_name;
     name_buf.length = strlen(name_buf.value) + 1;
     maj_stat = gss_import_name(&min_stat, &name_buf,
               (gss_OID) GSS_C_NT_HOSTBASED_SERVICE, &server_name);
     if (maj_stat != GSS_S_COMPLETE) {
          display_status("importing name", maj_stat, min_stat);
          if (mechOid != GSS_C_NO_OID)
                gss_release_oid(&min_stat, &mechOid);
          return -1;
     }
```

   今度も標準でない関数 gss_release_oid() を使用していることに注意してください。

   入力は、*name_buf* の文字列として指定されたサービス名です。出力は、gss_name_t 構造体 *server_name* へのポインタです。3 番目の引数

GSS_C_NT_HOSTBASED_SERVICE は *name_buf* に格納されている文字列の名前型です。この場合、文字列が *service@host* というサービスの形式で解釈されることを示します。

これらの作業が完了すると、サーバープログラムは次の gss_acquire_cred() を呼び出せます。

```
maj_stat = gss_acquire_cred(&min_stat, server_name, 0,
                                desiredMechs, GSS_C_ACCEPT,
                                server_creds, NULL, NULL);
```

- *min_stat* は関数から戻されるエラーコードです。
- *server_name* はサーバーの名前です。
- 0 は、プログラムが資格の有効期間の最大値を知る必要がないことを示します。
- *desiredMechs* は、この資格が適用される機構のセットです。
- GSS_C_ACCEPT は、資格がセキュリティーコンテキストを受け入れるためだけに使用できることを示します。
- *server_creds* は関数から戻される資格ハンドルです。
- NULL は、適用される機構や資格の有効期間をプログラムが知る必要がないことを示します。

server_acquire_creds() 関数のソースコードを、次に示します。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

例 6–2 server_acquire_creds() 関数のコード例

```
/*
 * Function: server_acquire_creds
 *
 * Purpose: imports a service name and acquires credentials for it
 *
 * Arguments:
 *
 *      service_name    (r) the ASCII service name
        mechType        (r) the mechanism type to use
 *      server_creds    (w) the GSS-API service credentials
 *
 * Returns: 0 on success, -1 on failure
 *
 * Effects:
 *
 * The service name is imported with gss_import_name, and service
 * credentials are acquired with gss_acquire_cred.  If either operation
 * fails, an error message is displayed and -1 is returned; otherwise,
 * 0 is returned.
 */
```

例 6–2 server_acquire_creds() 関数のコード例 (続き)

```
int server_acquire_creds(service_name, mechOid, server_creds)
     char *service_name;
     gss_OID mechOid;
     gss_cred_id_t *server_creds;
{
     gss_buffer_desc name_buf;
     gss_name_t server_name;
     OM_uint32 maj_stat, min_stat;
     gss_OID_set_desc mechOidSet;
     gss_OID_set desiredMechs = GSS_C_NULL_OID_SET;

     if (mechOid != GSS_C_NULL_OID) {
               desiredMechs = &mechOidSet;
               mechOidSet.count = 1;
               mechOidSet.elements = mechOid;
     } else
               desiredMechs = GSS_C_NULL_OID_SET;

     name_buf.value = service_name;
     name_buf.length = strlen(name_buf.value) + 1;
     maj_stat = gss_import_name(&min_stat, &name_buf,
               (gss_OID) GSS_C_NT_HOSTBASED_SERVICE, &server_name);
     if (maj_stat != GSS_S_COMPLETE) {
          display_status("importing name", maj_stat, min_stat);
          if (mechOid != GSS_C_NO_OID)
               gss_release_oid(&min_stat, &mechOid);
          return -1;
     }

     maj_stat = gss_acquire_cred(&min_stat, server_name, 0,
                                 desiredMechs, GSS_C_ACCEPT,
                                 server_creds, NULL, NULL);

     if (maj_stat != GSS_S_COMPLETE) {
          display_status("acquiring credentials", maj_stat, min_stat);
          return -1;
     }

     (void) gss_release_name(&min_stat, &server_name);

     return 0;
}
```

inetd の検査

サービスの資格の獲得が完了すると、gss-server は、inetd をユーザーが指定しているかどうかを検査します。main 関数が、次のようにして inetd の検査を行なっています。

```
if (do_inetd) {
     close(1);
     close(2);
```

inetd を使用するようにユーザーが指定している場合、プログラムは標準出力と標準エラーを閉じます。次に、gss-server は、inetd が接続の受け渡しに使用する標準入力を指定して sign_server() を呼び出します。それ以外の場合、gss-server は、ソケットを作成し、TCP 関数 accept() を使ってそのソケットの接続を受け入れたあと、accept() の戻り値のファイル記述子を指定して sign_server() を呼び出します。

inetd を使用しない場合、プログラムは終了されるまで接続とコンテキストを作成します。しかし、ユーザーが -once オプションを指定している場合、ループは最初の接続の後で終了します。

クライアントからのデータの受信

inetd の検査後、gss-server プログラムは、プログラムの主な作業を担う sign_server() を呼び出します。sign_server() はまず、server_establish_context() を呼び出してコンテキストを確立します。

sign_server() は次の作業を実行します。

- コンテキストを受け入れる
- データをラップ解除する
- データに署名する
- データを戻す

以降では、これらの作業について順次説明します。sign_server() 関数のソースコードを、次に示します。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

例 6–3 sign_server() 関数

```
int sign_server(s, server_creds)
     int s;
     gss_cred_id_t server_creds;
{
     gss_buffer_desc client_name, xmit_buf, msg_buf;
     gss_ctx_id_t context;
     OM_uint32 maj_stat, min_stat;
     int i, conf_state, ret_flags;
     char    *cp;

     /* Establish a context with the client */
     if (server_establish_context(s, server_creds, &context,
                    &client_name, &ret_flags) < 0)
     return(-1);

     printf("Accepted connection: \"%.*s\"\n",
```

例 6–3 sign_server() 関数 (続き)

```
        (int) client_name.length, (char *) client_name.value);
    (void) gss_release_buffer(&min_stat, &client_name);

    for (i=0; i < 3; i++)
        if (test_import_export_context(&context))
            return -1;

    /* Receive the sealed message token */
    if (recv_token(s, &xmit_buf) < 0)
        return(-1);

    if (verbose && log) {
        fprintf(log, "Sealed message token:\n");
        print_token(&xmit_buf);
    }

    maj_stat = gss_unwrap(&min_stat, context, &xmit_buf, &msg_buf,
                &conf_state, (gss_qop_t *) NULL);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("unsealing message", maj_stat, min_stat);
        return(-1);
    } else if (! conf_state) {
        fprintf(stderr, "Warning!  Message not encrypted.\n");
    }

    (void) gss_release_buffer(&min_stat, &xmit_buf);

    fprintf(log, "Received message: ");
    cp = msg_buf.value;
    if ((isprint(cp[0]) || isspace(cp[0])) &&
        (isprint(cp[1]) || isspace(cp[1]))) {
        fprintf(log, "\"%.*s\"\n", msg_buf.length, msg_buf.value);
    } else {
        printf("\n");
        print_token(&msg_buf);
    }

    /* Produce a signature block for the message */
    maj_stat = gss_get_mic(&min_stat, context, GSS_C_QOP_DEFAULT,
                &msg_buf, &xmit_buf);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("signing message", maj_stat, min_stat);
        return(-1);
    }

    (void) gss_release_buffer(&min_stat, &msg_buf);

    /* Send the signature block to the client */
    if (send_token(s, &xmit_buf) < 0)
        return(-1);

    (void) gss_release_buffer(&min_stat, &xmit_buf);

    /* Delete context */
    maj_stat = gss_delete_sec_context(&min_stat, &context, NULL);
    if (maj_stat != GSS_S_COMPLETE) {
```

例 6–3 sign_server() 関数 (続き)

```
        display_status("deleting context", maj_stat, min_stat);
        return(-1);
    }

    fflush(log);

    return(0);
}
```

コンテキストの受け入れ

一般に、コンテキストの確立時には、クライアントとサーバー間で一連のトークンが交換されます。プログラムの移植性を保つには、コンテキストの受け入れと起動の両方を、ループ内で実行する必要があります。コンテキスト受け入れループは、コンテキスト起動ループと非常によく似ています (ある意味で逆ですが)。103 ページの「サーバーとのセキュリティーコンテキストの確立」と比較してみてください。

server_establish_context() 関数のソースコードを、次に示します。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

例 6–4 server_establish_context() 関数

```
/*
 * Function: server_establish_context
 *
 * Purpose: establishes a GSS-API context as a specified service with
 * an incoming client, and returns the context handle and associated
 * client name
 *
 * Arguments:
 *
 *      s               (r) an established TCP connection to the client
 *      service_creds   (r) server credentials, from gss_acquire_cred
 *      context         (w) the established GSS-API context
 *      client_name     (w) the client's ASCII name
 *
 * Returns: 0 on success, -1 on failure
 *
 * Effects:
 *
 * Any valid client request is accepted.  If a context is established,
 * its handle is returned in context and the client name is returned
 * in client_name and 0 is returned.  If unsuccessful, an error
 * message is displayed and -1 is returned.
 */
```

例 6–4 server_establish_context() 関数 (続き)

```
int server_establish_context(s, server_creds, context, client_name, ret_flags)
     int s;
     gss_cred_id_t server_creds;
     gss_ctx_id_t *context;
     gss_buffer_t client_name;
     OM_uint32 *ret_flags;
{
     gss_buffer_desc send_tok, recv_tok;
     gss_name_t client;
     gss_OID doid;
     OM_uint32 maj_stat, min_stat, acc_sec_min_stat;
     gss_buffer_desc    oid_name;

     *context = GSS_C_NO_CONTEXT;

     do {
          if (recv_token(s, &recv_tok) < 0)
               return -1;

          if (verbose && log) {
              fprintf(log, "Received token (size=%d): \n", recv_tok.length);
              print_token(&recv_tok);
          }

          maj_stat =
               gss_accept_sec_context(&acc_sec_min_stat,
                                      context,
                                      server_creds,
                                      &recv_tok,
                                      GSS_C_NO_CHANNEL_BINDINGS,
                                      &client,
                                      &doid,
                                      &send_tok,
                                      ret_flags,
                                      NULL,     /* ignore time_rec */
                                      NULL);    /* ignore del_cred_handle */

          (void) gss_release_buffer(&min_stat, &recv_tok);

          if (send_tok.length != 0) {
               if (verbose && log) {
                    fprintf(log,
                          "Sending accept_sec_context token (size=%d):\n",
                          send_tok.length);
                    print_token(&send_tok);
               }
               if (send_token(s, &send_tok) < 0) {
                    fprintf(log, "failure sending token\n");
                    return -1;
               }

               (void) gss_release_buffer(&min_stat, &send_tok);
          }
          if (maj_stat!=GSS_S_COMPLETE && maj_stat!=GSS_S_CONTINUE_NEEDED) {
               display_status("accepting context", maj_stat,
                              acc_sec_min_stat);
```

例 6–4 server_establish_context() 関数 (続き)

```
                if (*context == GSS_C_NO_CONTEXT)
                        gss_delete_sec_context(&min_stat, context,
                                               GSS_C_NO_BUFFER);
                return -1;
            }

            if (verbose && log) {
                if (maj_stat == GSS_S_CONTINUE_NEEDED)
                    fprintf(log, "continue needed...\n");
                else
                    fprintf(log, "\n");
                fflush(log);
            }
        } while (maj_stat == GSS_S_CONTINUE_NEEDED);

        /* display the flags */
        display_ctx_flags(*ret_flags);

        if (verbose && log) {
            maj_stat = gss_oid_to_str(&min_stat, doid, &oid_name);
            if (maj_stat != GSS_S_COMPLETE) {
                display_status("converting oid->string", maj_stat, min_stat);
                return -1;
            }
            fprintf(log, "Accepted connection using mechanism OID %.*s.\n",
                    (int) oid_name.length, (char *) oid_name.value);
            (void) gss_release_buffer(&min_stat, &oid_name);
        }

        maj_stat = gss_display_name(&min_stat, client, client_name, &doid);
        if (maj_stat != GSS_S_COMPLETE) {
             display_status("displaying name", maj_stat, min_stat);
             return -1;
        }
        maj_stat = gss_release_name(&min_stat, &client);
        if (maj_stat != GSS_S_COMPLETE) {
             display_status("releasing name", maj_stat, min_stat);
             return -1;
        }
        return 0;
}
```

sign_server() 関数は、コンテキストを受け入れる際に、次のソースコードを使って server_establish_context() を呼び出します。

```
/* Establish a context with the client */
     if (server_establish_context(s, server_creds, &context,
                  &client_name, &ret_flags) < 0)
    return(-1);
```

server_establish_context() 関数はまず、クライアントがコンテキスト起動中に送信したトークンを探します。GSS-API 自身はトークンの送受信を行わないため、これらの作業を行うにはプログラムが独自のルーチンを持つ必要があります。サーバーは、次のように recv_token() を使ってトークンを受信します。

```
do {
     if (recv_token(s, &recv_tok) < 0)
          return -1;
```

次に、server_establish_context() は、GSS-API 関数 gss_accept_sec_context() を次のように呼び出します。

```
maj_stat = gss_accept_sec_context(&min_stat,
                                  context,
                                  server_creds,
                                  &recv_tok,
                                  GSS_C_NO_CHANNEL_BINDINGS,
                                  &client,
                                  &doid,
                                  &send_tok,
                                  ret_flags,
                                  NULL,     /* ignore time_rec */
                                  NULL);    /* ignore del_cred_handle */
```

- *min_stat* は実際の機構から戻されるエラー状態です。
- *context* は確立されているコンテキストです。
- *server_creds* は提供するサービスに対する資格です (118 ページの「資格の獲得」を参照)。
- *recv_tok* は recv_token() でクライアントから受信したトークンです。
- GSS_C_NO_CHANNEL_BINDINGS はチャネルバインディングを使用しないことを示すフラグです (83 ページの「GSS-API におけるチャネルバインディングの使用」を参照)。
- *client* はクライアント名 (ASCII 文字) です。
- *oid* は機構です (OID 形式)。
- *send_tok* はクライアントに送信するトークンです。
- *ret_flags* は、コンテキストが特定のオプション (message-sequence-detection など) をサポートするかどうかを示すさまざまなフラグです。
- 2 つの NULL 引数は、コンテキストの有効期間と、サーバーがクライアントのプロキシとして動作できるかどうかを、プログラムが知る必要がないことを示しています。

gss_accept_sec_context() が *maj_stat* に GSS_S_CONTINUE_NEEDED を設定している限り、受け入れループは継続します (エラーの場合を除く)。*maj_stat* の値が GSS_S_CONTINUE_NEEDED でも GSS_S_COMPLETE でもない場合、問題が発生したことを示しており、ループは終了します。

クライアントに送り返すべきトークンが存在するかどうかに関係なく、gss_accept_sec_context() は *send_tok* の長さを表す正の値を返します。次のステップでは、送信すべきトークンの存在の有無を確認し、存在する場合はそのトークンを送信します。

```
     if (send_tok.length != 0) {
          . . .
          if (send_token(s, &send_tok) < 0) {
               fprintf(log, "failure sending token\n");
               return -1;
          }

          (void) gss_release_buffer(&min_stat, &send_tok);
          }
```

メッセージのラップ解除

コンテキストの受け入れ後、`sign_server()` は、クライアントから送信されてきたメッセージを受け取ります。GSS-API にはトークン受信用の関数は用意されていないため、このプログラムは次のように `recv_token()` 関数を使用しています。

```
if (recv_token(s, &xmit_buf) < 0)
     return(-1);
```

メッセージは暗号化されている可能性があるため、プログラムは GSS-API 関数 `gss_unwrap()` でメッセージをラップ解除します。

```
maj_stat = gss_unwrap(&min_stat, context, &xmit_buf, &msg_buf,
                           &conf_state, (gss_qop_t *) NULL);
     if (maj_stat != GSS_S_COMPLETE) {
        display_status("unwrapping message", maj_stat, min_stat);
        return(-1);
     } else if (! conf_state) {
        fprintf(stderr, "Warning!  Message not encrypted.\n");
     }

     (void) gss_release_buffer(&min_stat, &xmit_buf);
```

`gss_unwrap()` は、`recv_token()` が *xmit_buf* に格納したメッセージを入力として受け取り、そのメッセージを変換し、その結果を *msg_buf* に格納します。`gss_unwrap()` への 2 つの引数に注目してください。*conf_state* は、このメッセージに機密性 (つまり暗号化) が適用されているかどうかを示すフラグです。最後の NULL は、メッセージ保護に使用された QOP をプログラムが知る必要がないことを示します。

メッセージへの署名とメッセージの返送

この時点で、`sign_server()` 関数はメッセージに署名する必要があります。メッセージへの署名には、メッセージのメッセージ整合性コード (MIC) のクライアントへの返送が伴います。メッセージを返送することで、メッセージの送信とラップ解除が正常に完了したことをクライアントに証明できます。MIC を取得するために、`sign_server()` は関数 `gss_get_mic()` を使用します。

```
maj_stat = gss_get_mic(&min_stat, context, GSS_C_QOP_DEFAULT,
                            &msg_buf, &xmit_buf);
```

gss_get_mic() は、*msg_buf* 内のメッセージに基づいて MIC を生成し、その結果を *xmit_buf* に格納します。次に、サーバーは send_token() で MIC をクライアントに返送します。クライアントは、gss_verify_mic() でその MIC を検証します。112 ページの「GSS-API クライアントにおける署名ブロックの読み取りと検証」を参照してください。

最後に、sign_server() はいくつかのクリーンアップを実行します。sign_server() は、gss_release_buffer() で GSS-API バッファーの *msg_buf* と *xmit_buf* を解放します。続いて、sign_server() は、gss_delete_sec_context() でコンテキストを破棄します。

test_import_export_context() 関数の使用

GSS-API を使用すると、コンテキストをエクスポートおよびインポートできます。これにより、マルチプロセスプログラムの異なるプロセス間でコンテキストを共有できます。sign_server() には概念検証用の関数 test_import_export_context() があります。この関数は、コンテキストのエクスポートとインポートがどのように機能するかを示します。test_import_export_context() は、コンテキストをプロセス間で渡すわけではありません。test_import_export_context() は、コンテキストをエクスポートするのにかかった時間を表示し、次に、インポートするのにかかった時間を表示します。test_import_export_context() は、実際には機能しない関数ですが、GSS-API のインポートおよびエクスポート機能をどのように使えばよいかを示しています。また、test_import_export_context() は、コンテキスト操作時のタイムスタンプの使い方も示しています。

test_import_export_context() のソースコードを、次に示します。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

例 6–5 test_import_export_context()

```
int test_import_export_context(context)
        gss_ctx_id_t *context;
{
        OM_uint32       min_stat, maj_stat;
        gss_buffer_desc context_token, copied_token;
        struct timeval tm1, tm2;

        /*
         * Attempt to save and then restore the context.
         */
        gettimeofday(&tm1, (struct timezone *)0);
        maj_stat = gss_export_sec_context(&min_stat, context, &context_token);
        if (maj_stat != GSS_S_COMPLETE) {
                display_status("exporting context", maj_stat, min_stat);
```

例 6–5 test_import_export_context() (続き)

```
                return 1;
        }
        gettimeofday(&tm2, (struct timezone *)0);
        if (verbose && log)
                fprintf(log, "Exported context: %d bytes, %7.4f seconds\n",
                        context_token.length, timeval_subtract(&tm2, &tm1));
        copied_token.length = context_token.length;
        copied_token.value = malloc(context_token.length);
        if (copied_token.value == 0) {
            fprintf(log, "Couldn't allocate memory to copy context token.\n");
            return 1;
        }
        memcpy(copied_token.value, context_token.value, copied_token.length);
        maj_stat = gss_import_sec_context(&min_stat, &copied_token, context);
        if (maj_stat != GSS_S_COMPLETE) {
                display_status("importing context", maj_stat, min_stat);
                return 1;
        }
        free(copied_token.value);
        gettimeofday(&tm1, (struct timezone *)0);
        if (verbose && log)
                fprintf(log, "Importing context: %7.4f seconds\n",
                        timeval_subtract(&tm1, &tm2));
        (void) gss_release_buffer(&min_stat, &context_token);
        return 0;
}
```

GSS-API サーバー例のクリーンアップ

main() 関数に戻ると、アプリケーションは gss_release_cred() でサービスの資格を削除します。機構の OID が指定された場合、プログラムは、gss_release_oid() でその OID を削除したあと、実行を終了します。

```
(void) gss_release_cred(&min_stat, &server_creds);
```

◆ ◆ ◆ 第 7 章

7 SASL を使用するアプリケーションの記述

SASL (Simple Authentication and Security Layer、簡易認証セキュリティー層) は、セキュリティーフレームワークの 1 つです。SASL (「ササル」と発音) は、接続ベースのプロトコルに対して認証サービスを提供するほか、オプションで整合性サービスと機密性サービスも提供します。この章で扱う内容は、次のとおりです。

簡易認証セキュリティー層 (SASL) の紹介

SASL は、アプリケーションと共有ライブラリの開発者に対し、認証、データ整合性検査、および暗号化を行うための機構を提供します。SASL を使用すると、開発者は汎用的な API に基づいたコーディングを行えます。これにより、特定の機構への依存を回避できます。SASL は特に、プロトコル IMAP、SMTP、ACAP、および LDAP を使用するアプリケーションに適しています。というのも、これらのプロトコルはすべて SASL をサポートしているからです。SASLは RFC 2222 に記述されています。

SASL ライブラリの基本

SASL ライブラリは `libsasl` と呼ばれます。`libsasl` は、正しく記述された SASL コンシューマアプリケーションがシステム上で利用可能な任意の SASL プラグインを使用できるようにするためのフレームワークです。「プラグイン」という用語は、SASL にサービスを提供するオブジェクトを指すために使用されます。プラグインは `libsasl` の外部に存在します。SASL プラグインを使用すれば、認証およびセキュリティー保護、名前の標準化、および補助プロパティー (パスワードなど) の検索を行えます。暗号化アルゴリズムは `libsasl` 内にではなくプラグイン内に格納されます。

`libsasl` は、コンシューマ (アプリケーションとライブラリ) に対してアプリケーションプログラミングインタフェース (API) を提供します。サービスプロバイダ

インタフェース (SPI) は、プラグインが libsasl にサービスを提供するために用意されたものです。libsasl はネットワークやプロトコルを認識しません。したがって、クライアントとサーバー間でデータの送受信を行うのは、アプリケーションの責任です。

SASL はユーザーに対して 2 つの重要な識別子を使用します。「認証 ID」(authid) は、ユーザーを認証するためのユーザー ID です。認証 ID は、システムへのアクセスをユーザーに許可します。「承認 ID」(userid) は、ユーザーが特定のオプションの使用を許可されているかどうかを検査する際に使用されます。

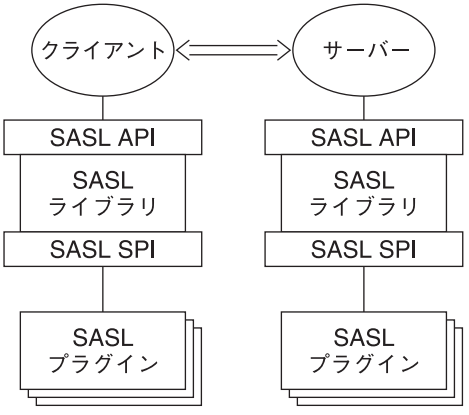
SASL クライアントアプリケーションと SASL サーバーアプリケーションは、共通の SASL 機構とセキュリティーレベルの折衝を行います。 通常の場合、SASL サーバーアプリケーションが、自身が受け入れ可能な認証機構のリストをクライアントに送信します。その後、SASL クライアントアプリケーションは、自身の要求にもっとも合う認証機構を決定できます。この時点から、両者で合意した認証機構に基づいて、「クライアントとサーバー間における一連の SASL 認証データの交換」として認証が実行されます。この交換は、認証に成功または失敗するか、あるいはクライアントかサーバーによって処理が中止されるまで継続されます。

認証処理中に、SASL 認証機構はセキュリティー層の折衝を行えます。セキュリティー層が選択された場合、SASL セッションが継続する限りその層を使用する必要があります。

SASL アーキテクチャー

次の図に、SASL の基本アーキテクチャーを示します。

図 7–1 SASL アーキテクチャー



クライアントアプリケーションとサーバーアプリケーションはそれぞれ、libsasl のローカルコピーに対して SASL API 経由で呼び出しを行います。libsasl は、SASL サービスプロバイダインタフェース (SPI) 経由で SASL 機構と通信します。

セキュリティー機構

セキュリティー機構プラグインは、libsasl にセキュリティーサービスを提供します。セキュリティー機構が提供する一般的な機能のいくつかを、次に示します。

- クライアント側での認証
- サーバー側での認証
- 整合性 (転送データが変更されていないことの検査)
- 機密性 (転送データの暗号化と復号化)

SASL セキュリティー強度係数 (SSF)

「SSF」(セキュリティー強度係数) は、SASL 保護の強さを示します。機構がセキュリティー層をサポートしている場合、クライアントとサーバーは SSF の折衝を行います。SSF の値は、SASL 折衝開始前に指定されたセキュリティープロパティーに基づいて決められます。折衝で 0 以外の SSF が決定された場合、認証完了時にクライアントとサーバーの両方でその機構のセキュリティー層を使用する必要があります。SSF は次のような整数値で表現されます。

- 0 – 保護なし。
- 1 – 整合性検査のみ。
- >1 – 認証、整合性、および機密性のサポート。その数値は暗号化の鍵の長さを表します。

機密性と整合性の処理はセキュリティー機構によって実行されます。libsasl はそれらの要求を調整する役割を果たします。

注 – SASL クライアントは折衝時に、最大の SSF を持つ機構を選択します。ただし、実際に選択された SASL 機構はその後、それよりも低い SSF の折衝を行う可能性があります。

SASL における通信

アプリケーションは libsasl API 経由で libsasl と通信します。libsasl はアプリケーションによって登録されたコールバックを介して追加情報の要求を行えます。アプリケーションがプラグインを直接呼び出すことはなく、必ず libsasl 経由でプラグインを呼び出します。プラグインは通常、libsasl フレームワークのプラグインを呼び出します。すると、フレームワークがアプリケーションのコールバックを呼び出します。SASL プラグインはアプリケーションを直接呼び出すことも可能です。ただしその際、アプリケーションは、プラグインからの呼び出しと libsasl からの呼び出しを区別できません。

コールバックは、次のようにさまざまな分野で役に立ちます。

- libsasl はコールバックを使用することで、認証を完了するのに必要な情報を取得できます。
- libsasl コンシューマアプリケーションはコールバックを使用することで、プラグインや構成データの検索パスを変更したり、ファイルを検証したり、さまざまなデフォルト動作を変更したりできます。
- サーバーはコールバックを使用することで、承認ポリシーを変更したり、異なるパスワード検証方式を提供したり、パスワード変更情報を取得したりできます。
- クライアントとサーバーはコールバックを使用することで、エラーメッセージの言語を指定できます。

アプリケーションが登録するコールバックには、 大域とセッションの 2 種類があります。さらに、libsasl では多数のコールバック ID が定義されており、それらの ID を使ってさまざまな種類のコールバックを登録できるようになっています。特定の種類のコールバックが登録されていない場合、libsasl はデフォルトの動作を実行します。

セッションコールバックは大域コールバックよりも優先されます。ある ID に対してセッションコールバックが指定された場合、そのセッションでは大域コールバックは呼び出されません。コールバックによっては、大域でなければなりません。これは、それらのコールバックがセッションの外側で呼び出されるためです。次のような処理は大域コールバックにする必要があります。

- プラグイン読み込み時の検索パスの決定
- プラグインの検証
- 構成データの検索
- エラーメッセージのロギング
- libsasl またはプラグインに関するその他の大域構成

特定の SASL コールバック ID の SASL コールバックとして、NULL コールバック関数を登録できます。NULL コールバック関数は、必要なデータを提供する準備がクライアント側に整っていることを示します。SASL コールバック ID には必ず、接頭辞 SASL_CB_ が付いています。

SASL が提供する次のコールバックは、クライアントまたはサーバーで使用できます。

SASL_CB_GETOPT	SASL オプションを取得します。オプションを設定すると、libsasl(3LIB) とその関連プラグインの動作が変更されます。クライアントまたはサーバーで使用できます。
SASL_CB_LOG	libsasl とそのプラグインに対するロギング機能を設定します。デフォルトの動作は「syslog の使用」です。
SASL_CB_GETPATH	SASL プラグイン検索パスのコロン区切りリストを取得します。デフォルトのパスは次のとおりです。

- 32 ビット SPARC アーキテクチャー: `/usr/lib/sasl`
- 32 ビット x86 アーキテクチャー: `/usr/lib/sasl`
- 64 ビット SPARC アーキテクチャー: `/usr/lib/sasl/sparcv9`
- x64 アーキテクチャー: `/usr/lib/sasl/amd64`

SASL_CB_GETCONF　SASL サーバーの構成ディレクトリへのパスを取得します。デフォルトは `/etc/sasl` です。

SASL_CB_LANGUAGE　優先順に並んだ RFC 1766 言語コードのコンマ区切りリストを指定します。このリストは、クライアントおよびサーバーのエラーメッセージとクライアントのプロンプトで使用されます。デフォルトは `i-default` です。

SASL_CB_VERIFYFILE　構成ファイルとプラグインファイルを検証します。

SASL が提供する次のコールバックは、クライアント専用です。

SASL_CB_USER　クライアントのユーザー名を取得します。このユーザー名は承認 ID と同一です。LOGNAME 環境変数がデフォルトになります。

SASL_CB_AUTHNAME　クライアントの認証名を取得します。

SASL_CB_PASS　クライアントのパスフレーズベースのパスワードを取得します。

SASL_CB_ECHOPROMPT　指定されたチャレンジプロンプトの結果を取得します。クライアントからの入力をエコーできます。

SASL_CB_NOECHOPROMPT　指定されたチャレンジプロンプトの結果を取得します。クライアントからの入力をエコーするべきではありません。

SASL_CB_GETREALM　認証に使用するレルムを設定します。

SASL が提供する次のコールバックは、サーバー専用です。

SASL_CB_PROXY_POLICY
: 認証されたユーザーが指定されたユーザーの代理役として承認されているかどうかを検査します。このコールバックが登録されていない場合、認証されたユーザーと承認されるべきユーザーが同一でなければなりません。これらの ID が同一でない場合、認証が失敗します。標準でない承認ポリシーを処理するには、サーバーアプリケーションを使用してください。

SASL_CB_SERVER_USERDB_CHECKPASS
: 呼び出し元から提供されたユーザーデータベースに対して平文パスワードを検証します。

SASL_CB_SERVER_USERDB_SETPASS
　ユーザーデータベース内に平文パスワードを格納します。

SASL_CB_CANON_USER
　アプリケーションから提供されたユーザー標準化関数を呼び出します。

最初の SASL ライブラリ初期化時に、サーバーとクライアントは必要な大域コールバックのすべてを宣言します。大域コールバックが利用可能なのは、SASL セッション初期化前と SASL セッション中です。セッション初期化前には、プラグインの読み込み、データのロギング、構成ファイルの読み取りなどの作業がコールバックによって実行されます。SASL セッションの開始時に、追加のコールバックを宣言できます。そうしたコールバックは、必要に応じて大域コールバックを上書きできます。

SASL 接続コンテキスト

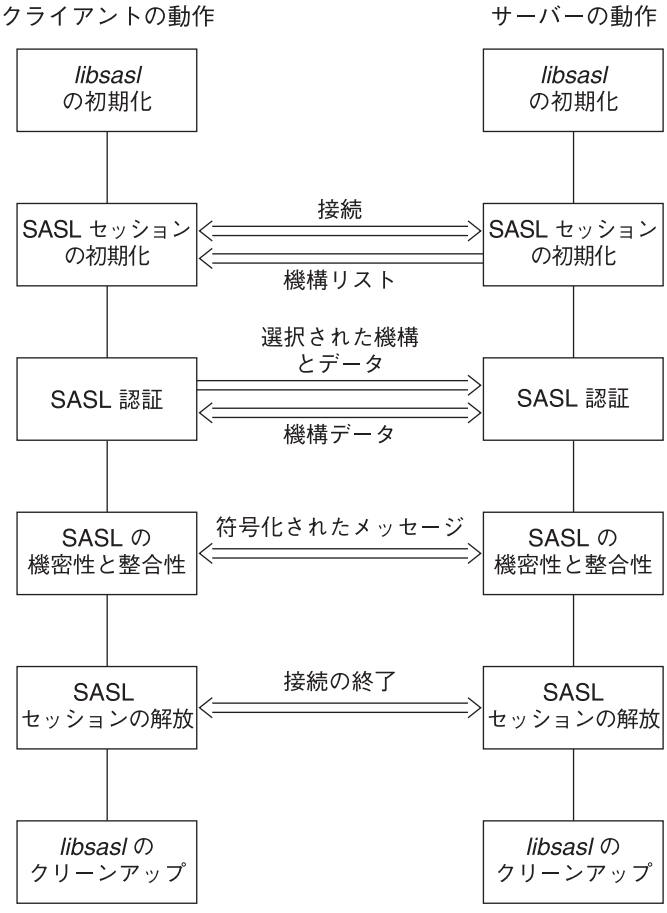
`libsasl` は、SASL 接続の「コンテキスト」内に、SASL クライアントと SASL サーバーの両方に対する各 SASL セッションの状態を格納します。各コンテキストは、同時に 1 つの認証および 1 つのセキュリティーセッションでしか使用できません。格納される状態としては、次のような情報があります。

- サービス、名前情報およびアドレス情報、プロトコルフラグなどの接続情報
- 接続に固有のコールバック
- SASL SSF の折衝に使われるセキュリティープロパティー
- 認証の状態とセキュリティー層の情報

SASL サイクル内のステップ

次の図に、SASL ライフサイクル内に含まれる各種ステップを示します。クライアントの動作は図の左側に、サーバーの動作は右側に、それぞれ示してあります。その間に描かれた矢印は、外部接続経由でのクライアントとサーバー間の相互作用を示しています。

図 7–2 SASL ライフサイクル



次からの節で、このライフサイクル内のステップについて説明します。

libsasl の初期化

クライアントは sasl_client_init() を呼び出すことで libsasl をクライアント用に初期化します。サーバーは、sasl_server_init() を呼び出すことで libsasl をサーバー用に初期化します。

sasl_client_init() の実行時には、SASL クライアントプラグイン、クライアントの機構プラグイン、およびクライアントの標準化プラグインが読み込まれます。同様に、sasl_server_init() の呼び出し時には、SASL サーバープラグイン、サーバーの機構プラグイン、サーバーの標準化プラグイン、およびサーバーの auxprop プラグインが読み込まれます。sasl_client_init() の呼び出し後、追加のクライアントプラグインは、sasl_client_add_plugin() と sasl_canonuser_add_plugin() を使用して追

加できます。サーバー側では、sasl_server_init() の呼び出し後、追加のサーバープラグインは、sasl_server_add_plugin()、sasl_canonuser_add_plugin()、および sasl_auxprop_add_plugin() を使用して追加できます。SASL 機構は Solaris ソフトウェアの次のディレクトリに格納されます。

- 32 ビット SPARC アーキテクチャー: /usr/lib/sasl
- 32 ビット x86 アーキテクチャー: /usr/lib/sasl
- 64 ビット SPARC アーキテクチャー: /usr/lib/sasl/sparcv9
- x64 アーキテクチャー: /usr/lib/sasl/amd64

ただし、SASL_CB_GETPATH コールバックを使えば、このデフォルトの場所を変更できます。

この時点で、必要な大域コールバックのすべてが設定されています。SASL クライアントと SASL サーバーには、次のコールバックを含めることができます。

- SASL_CB_GETOPT
- SASL_CB_LOG
- SASL_CB_GETPATH
- SASL_CB_VERIFYFILE

さらに SASL サーバーには、SASL_CB_GETCONF コールバックも含めることができます。

SASL セッションの初期化

サーバーとクライアントはプロトコル経由で接続を確立します。SASL による認証を行う場合、サーバーとクライアントは SASL 接続コンテキストを作成します。それには、sasl_server_new() と sasl_client_new() をそれぞれ使用します。SASL クライアントと SASL サーバーは、sasl_setprop() を使って機構に対するセキュリティー制約プロパティーを設定できます。これにより、SASL コンシューマアプリケーションは、指定された SASL 接続コンテキストの最小 SSF、最大 SSF、およびセキュリティープロパティーを決定できます。

```
#define SASL_SEC_NOPLAINTEXT            0x0001
#define SASL_SEC_NOACTIVE               0x0002
#define SASL_SEC_NODICTIONARY           0x0004
#define SASL_SEC_FORWARD_SECRECY        0x0008
#define SASL_SEC_NOANONYMOUS            0x0010
#define SASL_SEC_PASS_CREDENTIALS       0x0020
#define SASL_SEC_MUTUAL_AUTH            0x0040
```

注 – 認証とセキュリティー層は、クライアント/サーバー間のプロトコルや libsasl 以外の機構を使って提供してもかまいません。そうした場合、sasl_setprop() 経由で外部認証 ID や外部 SSF を設定できます。たとえば、プロトコルが SSL を使用してサーバーに対するクライアント認証を行う場合を考えます。この場合、外部認証 ID をクライアントの被認証者名として、外部 SSF を鍵のサイズとして、それぞれ使用できます。

サーバー側では、libsasl が、セキュリティープロパティーと外部 SSF に従って利用可能な SASL 機構を決定します。クライアントは、その利用可能な SASL 機構を SASL サーバーからプロトコル経由で取得します。

SASL サーバー側で SASL 接続コンテキストを作成するには、sasl_server_new() を呼び出す必要があります。すでに使われていない既存の SASL 接続コンテキストを再利用することも可能です。ただし、その場合は次のパラメータをリセットする必要があります。

```
#define SASL_DEFUSERREALM 3     /* default realm passed to server_new or set with setprop */
#define SASL_IPLOCALPORT 8      /* iplocalport string passed to server_new */
#define SASL_IPREMOTEPORT 9     /* ipremoteport string passed to server_new */
#define SASL_SERVICE    12      /* service passed to sasl_*_new */
#define SASL_SERVERFQDN 13      /* serverFQDN passed to sasl_*_new */
```

sasl_client_new() と sasl_server_new() に対するパラメータは、コールバックとプロトコルフラグ以外はすべて変更可能です。

また、サーバーとクライアントは、セキュリティーポリシーの確立や接続固有パラメータの設定も行えます。それには、sasl_setprop() を使って次のプロパティーを指定します。

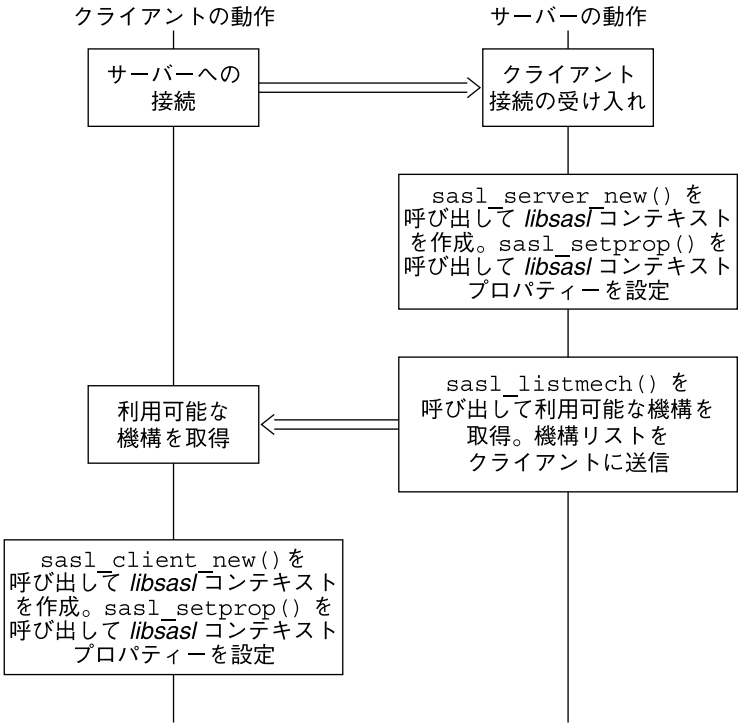
```
#define SASL_SSF_EXTERNAL 100 /* external SSF active (sasl_ssf_t *) */
#define SASL_SEC_PROPS 101 /* sasl_security_properties_t */
#define SASL_AUTH_EXTERNAL 102 /* external authentication ID (const char *)
 */
```

- SASL_SSF_EXTERNAL – 強度係数 (鍵のビット数) の設定用
- SASL_SEC_PROPS – セキュリティーポリシーの定義用
- SASL_AUTH_EXTERNAL – 外部認証 ID

サーバーは、sasl_listmech() を呼び出すことで、セキュリティーポリシーを満たす利用可能な SASL 機構のリストを取得できます。クライアントは通常、利用可能な機構リストをプロトコルに依存した方法でサーバーから取得できます。

SASL セッションの初期化を図示したのが、次の図です。この図と後続の図では、プロトコル経由でデータを転送した後の検査処理は、図を単純化する目的で省略しています。

図 7–3 SASL セッションの初期化



SASL 認証

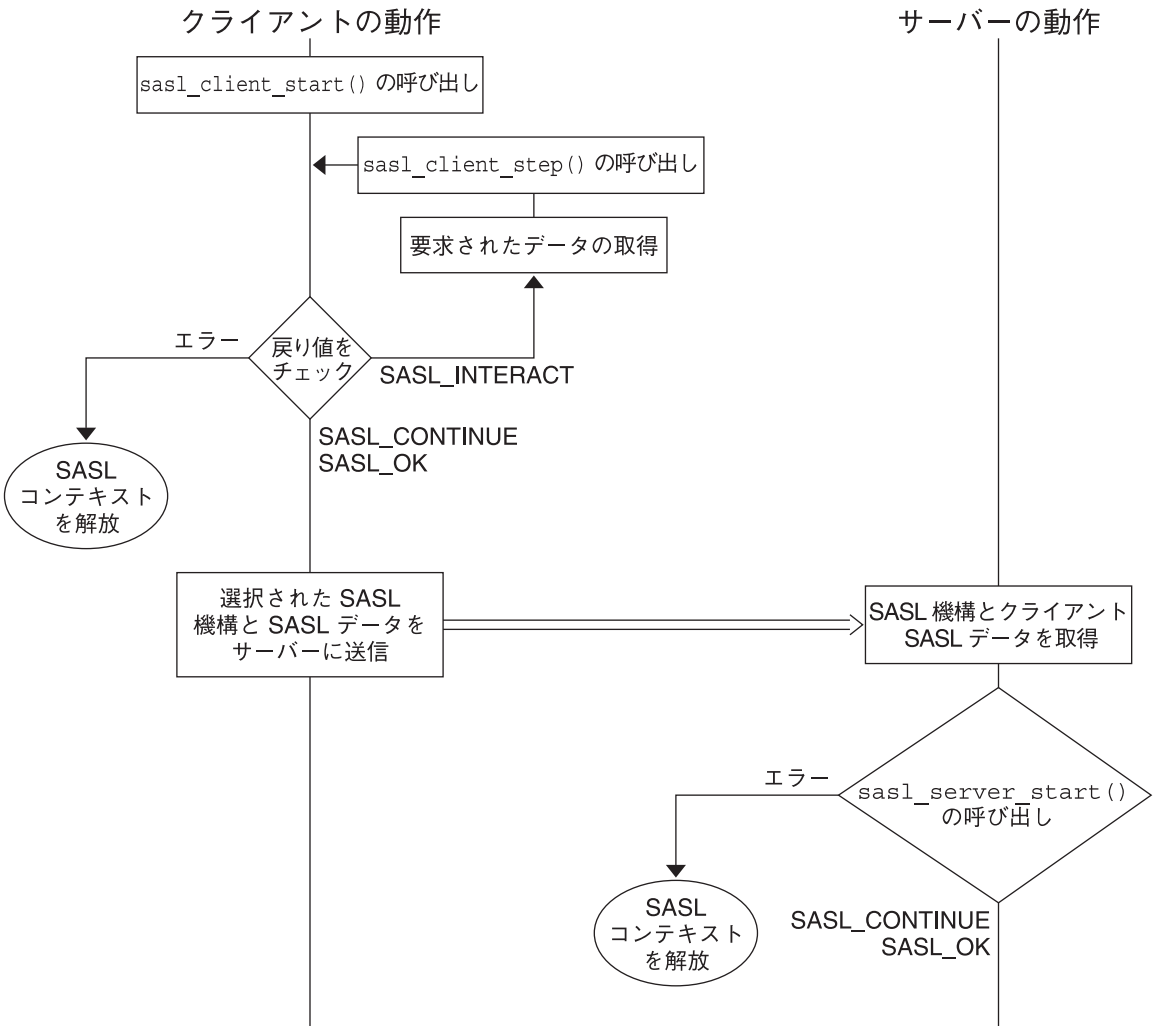
認証時にクライアントとサーバーで実行されるステップの数は、使用されるセキュリティー機構ごとに異なります。SASL クライアントは、使用すべきセキュリティー機構のリストを指定して `sasl_client_start()` を呼び出します。このリストは通常、サーバーから送られてきます。`libsasl` は、利用可能な機構とクライアントのセキュリティーポリシーに基づいて、この SASL セッションに最適な機構を選択します。クライアントのセキュリティーポリシーは、許可される機構を制御します。そして、選択された機構が `sasl_client_start()` から返されます。クライアントのセキュリティー機構は、認証時に追加情報を必要とする場合があります。登録されたコールバック関数が NULL でない限り、`libsasl` はその指定されたコールバックを呼び出します。コールバック関数が NULL である場合、`libsasl` は、SASL_INTERACT と必要情報の要求を返します。SASL_INTERACT が返された場合、要求された情報を指定して `sasl_client_start()` を呼び出す必要があります。

`sasl_client_start()` から SASL_CONTINUE または SASL_OK が返された場合、クライアントは、選択された機構と結果の認証データを、サーバーに送信する必要があります。その他の値が返された場合、何らかのエラーが発生しています。たとえば、利用可能な機構が存在しない、などです。

サーバーは、クライアントによって選択された機構と、認証データを受信します。続いてサーバーは、sasl_server_start() を使ってこのセッション用に機構データを初期化します。また、sasl_server_start() は認証データの処理も行います。sasl_server_start() から SASL_CONTINUE または SASL_OK が返された場合、サーバーは認証データを送信します。sasl_server_start() からその他の値が返された場合、機構の受け入れに失敗した、認証に失敗した、など、何らかのエラーが発生しています。その認証は中止する必要があります。その SASL コンテキストは、解放するか、または再利用する必要があります。

認証プロセスのうち、以上の部分を図示したのが、次の図です。

図 7–4 SASL 認証: クライアントデータの送信

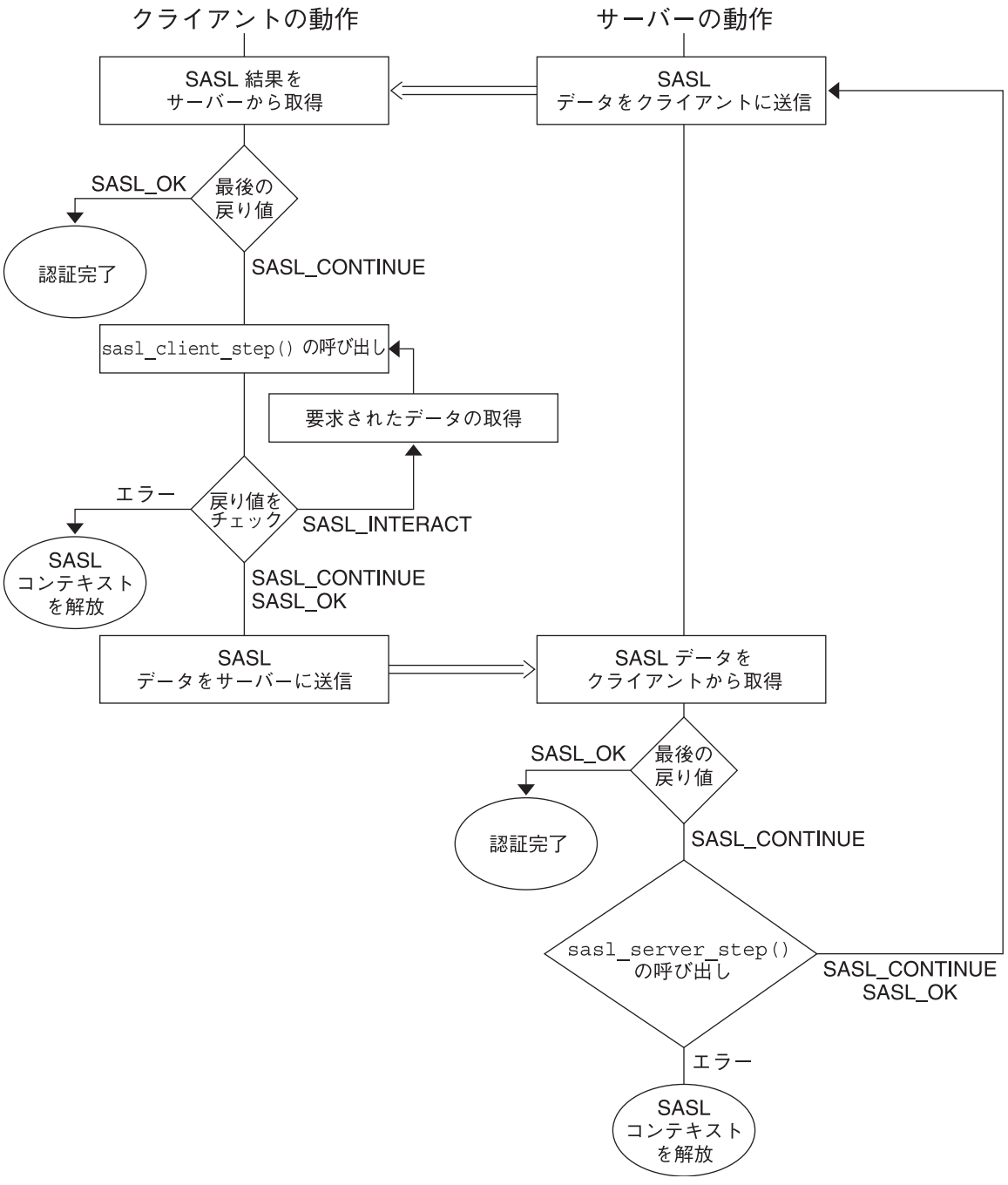


サーバー側の `sasl_server_start()` 呼び出しから SASL_CONTINUE が返された場合、サーバーは必要な認証情報をすべて取得するために、クライアントとの通信を継続します。後続のステップ数は機構ごとに異なります。必要に応じて、クライアントは `sasl_client_step()` を呼び出すことで、サーバーからの認証データを処理し、応答を生成します。同様に、サーバーは `sasl_server_step()` を呼び出すことで、クライアントからの認証データを処理し、応答を生成できます。この交換は、認証が完了するか、エラーが発生するまで継続されます。SASL_OK が返された場合、それはクライアント側またはサーバー側での認証が正常に完了したことを意味します。他方の認証を完了させるために他方に送信すべき追加データが、SASL 機

構内にまだ残っている可能性があります。サーバー側とクライアント側の両方で認証が完了すると、サーバーとクライアントは、互いのプロパティーを照会できるようになります。

次の図は、追加認証データを転送する際の、サーバーとクライアント間における相互作用を示したものです。

図 7–5 SASL 認証: サーバーデータの処理



SASL の機密性と整合性

セキュリティー層の有無を検査するには、sasl_getprop(3SASL) 関数を使ってセキュリティー強度係数 (SSF) の値が 0 よりも大きいかどうかを確認します。セキュリティー層の折衝が行われていた場合、クライアントとサーバーは認証成功後に結果の SSF を使用する必要があります。クライアントとサーバー間のデータ交換は、認証の場合と似た方法で行われます。プロトコルによってクライアントまたはサーバーにデータが送信される前に、データに sasl_encode() が適用されます。受信側では、最後に、sasl_decode() によってデータが復号化されます。セキュリティー層の折衝がなされていなかった場合、その SASL 接続コンテキストは必要ありません。したがって、このコンテキストは破棄または再利用してかまいません。

SASL セッションの解放

SASL 接続コンテキストを解放する必要があるのは、そのセッションを再利用しない場合だけです。sasl_dispose() は、SASL 接続コンテキストと関連するすべてのリソースおよび機構を解放します。sasl_done() を呼び出す場合、その前に SASL 接続コンテキストを破棄しておく必要があります。SASL 接続に対するコンテキストリソースを解放することは、sasl_done() の責任ではありません。145 ページの「libsasl のクリーンアップ」を参照してください。

SASL セッションが解放される際、すべての状態が解放される可能性がある旨が、関連する機構に通知されます。SASL セッションを解放する必要があるのは、そのセッションを再利用しない場合だけです。それ以外の場合、別のセッションがその SASL 状態を再利用できます。クライアントとサーバーのどちらも、sasl_dispose() を使って SASL 接続コンテキストを解放します。

libsasl のクリーンアップ

このステップでは、SASL ライブラリとプラグインのすべてのリソースを解放します。クライアントとサーバーは sasl_done() を呼び出すことで、libsasl() のリソースを解放し、すべての SASL プラグインを読み込み解除します。sasl_done() は SASL 接続コンテキストを解放しません。アプリケーションが SASL クライアントでもあり、かつ SASL サーバーでもある場合、sasl_done() によって SASL クライアントと SASL サーバー双方のリソースが解放される点に注意してください。クライアント、サーバーのいずれかのリソースのみを解放することはできません。

注意 – ライブラリ内で sasl_done() を呼び出すべきではありません。アプリケーション内で sasl_done() を呼び出す際には、libsasl を使用している可能性のあるすべてのライブラリとの干渉を回避できるように、細心の注意を払う必要があります。

SASL の例

ここでは、クライアントアプリケーションとサーバーアプリケーション間における一般的な SASL セッションを示します。この例の実行手順は次のとおりです。

1. クライアントアプリケーションは、`libsasl` を初期化し、次の大域コールバックを設定します。
   - SASL_CB_GETREALM
   - SASL_CB_USER
   - SASL_CB_AUTHNAME
   - SASL_CB_PASS
   - SASL_CB_GETPATH
   - SASL_CB_LIST_END
2. サーバーアプリケーションは、`libsasl` を初期化し、次の大域コールバックを設定します。
   - SASL_CB_LOG
   - SASL_CB_LIST_END
3. クライアントは、SASL 接続コンテキストを作成し、セキュリティープロパティーを設定し、利用可能な機構のリストをサーバーに要求します。
4. サーバーは、SASL 接続コンテキストを作成し、セキュリティープロパティーを設定し、適切な SASL 機構のリストを取得し、それをクライアントに送信します。
5. クライアントは、利用可能な機構のリストを取得し、特定の機構を選択し、その選択した機構と認証データをサーバーに送信します。
6. 続いて、クライアントとサーバーは、認証とセキュリティー層の折衝が完了するまで、SASL データを交換します。
7. 認証が完了すると、クライアントとサーバーは、セキュリティー層の折衝が行われたどうかを判断します。クライアントはテストメッセージを符号化します。そのメッセージがサーバーに送信されます。サーバーは、認証ユーザーのユーザー名とそのユーザーのレルムも決定します。
8. サーバーは、符号化されたメッセージを受信、復号化、および出力します。
9. クライアントは、`sasl_dispose()` を呼び出してクライアントの SASL 接続コンテキストを解放します。続いてクライアントは、`sasl_done()` を呼び出して `libsasl` のリソースを解放します。
10. サーバーは、`sasl_dispose()` を呼び出してそのクライアントの接続コンテキストを解放します。

次に、クライアントとサーバー間のやり取りの様子を示します。`libsasl` への呼び出しが発生するたびにその呼び出し内容が表示されています。送信側および受信側によるそれぞれのデータ転送も表示されています。なお、データは符号化された形式で表示され、その先頭に転送元が示されています。 転送元がクライアントの場合は

C:、サーバーの場合は S: と表示されます。両アプリケーションのソースコードについては、付録 D 「SASL ソースコード例」を参照してください。

クライアント

```
% doc-sample-client
*** Calling sasl_client_init() to initialize libsasl for client use ***
*** Calling sasl_client_new() to create client SASL connection context ***
*** Calling sasl_setprop() to set sasl context security properties ***
Waiting for mechanism list from server...
```

サーバー

```
% doc-sample-server digest-md5
*** Calling sasl_server_init() to initialize libsasl for server use ***
*** Calling sasl_server_new() to create server SASL connection context ***
*** Calling sasl_setprop() to set sasl context security properties ***
Forcing use of mechanism digest-md5
Sending list of 1 mechanism(s)
S: ZGlnZXN0LW1kNQ==
```

クライアント

```
                        S: ZGlnZXN0LW1kNQ==
                        received 10 byte message
                        got 'digest-md5'
                        Choosing best mechanism from: digest-md5
                        *** Calling sasl_client_start() ***
                        Using mechanism DIGEST-MD5
                        Sending initial response...
                        C: RElHRVNULU1ENQ==
                        Waiting for server reply...
```

サーバー

```
C: RElHRVNULU1ENQ==
got 'DIGEST-MD5'
*** Calling sasl_server_start() ***
Sending response...
S: bm9uY2U9IklicGxhRHJZNE4Z1gyVm5lQzl5MTZOYWxUOVcvanUrcmp5YmRqaHM\
sbT0iam0xMTQxNDIiLHFvcD0iYXV0aCxhdXRoLWludCxhdXRoLWNvbmYiLGNpcGhlcj0ic\
QwLHJjNC01NixyYzQiLG1heGJ1Zj0yMDQ4LGNoYXJzZXQ9dXRmLTgsYWxnb3JpdGhtPW1k\
XNz
Waiting for client reply...
```

クライアント

```
S: bm9uY2U9IklicGxhRHJZNE4Z1gyVm5lQzl5MTZOYWxUOVcvanUrcmp5YmRqaHM\
sbT0iam0xMTQxNDIiLHFvcD0iYXV0aCxhdXRoLWludCxhdXRoLWNvbmYiLGNpcGhlcj0ic\
QwLHJjNC01NixyYzQiLG1heGJ1Zj0yMDQ4LGNoYXJzZXQ9dXRmLTgsYWxnb3JpdGhtPW1k\
XNz
received 171 byte message
got 'nonce="IbplaDrY4N4szhgX2VneC9y16NalT9W/ju+rjybdjhs=",\
realm="jm114142",qop="auth,auth-int,auth-conf",cipher="rc4-40,rc4-56,\
rc4",maxbuf=2048,charset=utf-8,algorithm=md5-sess'
*** Calling sasl_client_step() ***
Please enter your authorization name : zzzz
Please enter your authentication name : zzzz
Please enter your password : zz
```

```
*** Calling sasl_client_step() ***
Sending response...
C: dXNlcm5hbWU9Inp6enoiLHJlYWxtPSJqbTExNDE0MiIsbm9uY2U9IklicGxhRHJZNE4\
yVm5lQzl5MTZOYWxUOVcvanUrcmp5YmRqaHM9Iixjbm9uY2U9InlqZ2hMVmhjRFJMa0Fob\
tDS0p2WVUxMUM4V1NycjJVWm5IR2Vkclk9IixuYz0wMDAwMDAwMSxxb3A9YXV0aC1jb25m\
Ghlcj0icmM0IixtYXhidWY9MjA0OCxkaWdlc3QtdXJpPSJyY21kLyIscmVzcG9uc2U9OTY\
ODI1MmRmNzY4YTJjYzkxYjJjZDMyYTk0ZWM=
Waiting for server reply...
```

サーバー

```
C: dXNlcm5hbWU9Inp6enoiLHJlYWxtPSJqbTExNDE0MiIsbm9uY2U9IklicGxhRHJZNE4\
yVm5lQzl5MTZOYWxUOVcvanUrcmp5YmRqaHM9Iixjbm9uY2U9InlqZ2hMVmhjRFJMa0Fob\
tDS0p2WVUxMUM4V1NycjJVWm5IR2Vkclk9IixuYz0wMDAwMDAwMSxxb3A9YXV0aC1jb25m\
Ghlcj0icmM0IixtYXhidWY9MjA0OCxkaWdlc3QtdXJpPSJyY21kLyIscmVzcG9uc2U9OTY\
ODI1MmRmNzY4YTJjYzkxYjJjZDMyYTk0ZWM=
got 'username="zzzz",realm="jm114142",\
nonce="IbplaDrY4N4szhgX2VneC9y16NalT9W/ju+rjybdjhs=",\
cnonce="yjghLVhcDRLkAhoirwKCKJvYU11C8WSrr2UZnHGedrY=", \
nc=00000001,qop=auth-conf,cipher="rc4",maxbuf=2048,digest-uri="rcmd/",\
response=966e978252df768a2cc91b2cd32a94ec'
*** Calling sasl_server_step() ***
Sending response...
S: cnNwYXV0aD0yYjEzMzRjYzU4NTE4MTEwOWM3OTdhMjUwYjkwMzk3OQ==
Waiting for client reply...
```

クライアント

```
S: cnNwYXV0aD0yYjEzMzRjYzU4NTE4MTEwOWM3OTdhMjUwYjkwMzk3OQ==
received 40 byte message
got 'rspauth=2b1334cc585181109c797a250b903979'
*** Calling sasl_client_step() ***
C:
Negotiation complete
*** Calling sasl_getprop() ***
Username: zzzz
SSF: 128
Waiting for encoded message...
```

サーバー

```
Waiting for client reply...
C: got '' *** Calling sasl_server_step() ***
Negotiation complete
*** Calling sasl_getprop() to get username, realm, ssf ***
Username: zzzz
Realm: 22c38
SSF: 128
*** Calling sasl_encode() *** sending encrypted message 'srv message 1'
S: AAAAHvArjnAvDFuMBqAAxkqdumzJB6VD1oajiwABAAAAAA==
```

クライアント

```
S: AAAAHvArjnAvDFuMBqAAxkqdumzJB6VD1oajiwABAAAAAA==
received 34 byte message
got ''
*** Calling sasl_decode() ***
received decoded message 'srv message 1'
*** Calling sasl_encode() ***
```

```
sending encrypted message 'client message 1'
C: AAAAIRdkTEMYOn9X4NXkxPc3OTFvAZUnLbZANqzn6gABAAAAAA==
*** Calling sasl_dispose() to release client SASL connection context ***
*** Calling sasl_done() to release libsasl resources ***
```

サーバー

```
Waiting for encrypted message...
C: AAAAIRdkTEMYOn9X4NXkxPc3OTFvAZUnLbZANqzn6gABAAAAAA==
got ''
*** Calling sasl_decode() ***
received decoded message 'client message 1'
*** Calling sasl_dispose() to release client SASL connection context ***
```

サービスプロバイダ用の SASL

ここでは、SASL アプリケーションに対して機構やその他のサービスを提供するプラグインの作成方法について説明します。

注 – エクスポート規則のため、Solaris SASL サービスプロバイダインタフェース (SPI) は、Solaris 以外のクライアントおよびサーバー機構プラグイン用のセキュリティーレイヤーをサポートしません。このため、Solaris 以外のクライアントおよびサーバー機構プラグインは、整合性サービスまたは機密性サービスを提供できません。Solaris のクライアントおよびサーバー機構プラグインには、この制限はありません。

SASL プラグインの概要

SASL サービスプロバイダインタフェース (SPI) を使うと、プラグインと `libsasl` ライブラリ間の通信が可能となります。SASL プラグインは通常、共有ライブラリとして実装されます。1 つの共有ライブラリには、各種の SASL プラグインを 1 つ以上格納できます。共有ライブラリ内のプラグインは、`libsasl` によって `dlopen(3C)` 関数で動的にオープンされます。

また、プラグインは、`libsasl` を呼び出す特定のアプリケーションに静的にバインドすることも可能です。こうした種類のプラグインを読み込むには、`sasl_client_add_plugin()` 関数または `sasl_server_add_plugin()` 関数を使用しますが、どちらを使用するかは、そのアプリケーションがクライアント、サーバーのいずれであるかによります。

Solaris オペレーティングシステム内の SASL プラグインは、次の要件を満たす必要があります。

- 共有ライブラリ内のプラグインは、有効な実行可能オブジェクトファイル (推奨のファイル拡張子は .so) 内に格納されている必要があります。
- プラグインは検証可能な場所に収められている必要があります。プラグインの検証には SASL_CB_VERIFYFILE コールバックが使用されます。
- プラグインは適切なエントリポイントを含んでいる必要があります。
- SASL クライアントのプラグインのバージョンが、SASL サーバーの対応するプラグインのバージョンに一致している必要があります。
- プラグインは正常に初期化できる必要があります。
- プラグインのバイナリタイプが、libsasl のバイナリタイプに一致している必要があります。

SASL プラグインは次の 4 つのカテゴリに分類されます。

- クライアント機構プラグイン
- サーバー機構プラグイン
- 標準化プラグイン
- auxprop プラグイン

sasl_client_init() 関数を呼び出すと、利用可能なすべてのクライアントプラグインが SASL クライアントに読み込まれます。 sasl_server_init() 関数を呼び出すと、サーバープラグイン、標準化プラグイン、および auxprop プラグインが SASL サーバーに読み込まれます。sasl_done() を呼び出すと、すべてのプラグインが読み込み解除されます。

libsasl はプラグイン検索時に、SASL_CB_GETPATH コールバック関数、デフォルトパスのいずれかを使用します。SASL_CB_GETPATH は、プラグイン検索ディレクトリのコロン区切りリストを返します。SASL コンシューマが SASL_CB_GETPATH コールバックを指定した場合、libsasl はそのコールバックから返されたパスを使って検索を行います。それ以外の場合、SASL コンシューマはバイナリタイプに対応するデフォルトパスを使用できます。

- 32 ビット SPARC アーキテクチャー: /usr/lib/sasl
- 32 ビット x86 アーキテクチャー: /usr/lib/sasl
- 64 ビット SPARC アーキテクチャー: /usr/lib/sasl/sparcv9
- x64 アーキテクチャー: /usr/lib/sasl/amd64

libsasl は読み込みプロセスの一部として、サポートされている最新バージョンのプラグインを呼び出します。そのプラグインは、バージョンと自身を記述した構造体を返します。バージョンが一致した場合、libsasl はそのプラグインを読み込みます。現在のバージョン番号 (SASL_UTILS_VERSION) は 4 です。

特定のプラグインの初期化が完了すると、そのプラグインと libsasl との間のそれ以降の通信に必要となる構造体を確立する必要があります。プラグイン

は、sasl_utils_t 構造体を使って libsasl を呼び出します。libsasl は、次の構造体に含まれるエントリポイントを使ってプラグインと通信します。

- sasl_out_params_t
- sasl_client_params_t
- sasl_server_params_t
- sasl_client_plug_t
- sasl_server_plug_t
- sasl_canonuser_plug_t
- sasl_auxprop_plug_t

これらの構造体のソースコードは、SASL ヘッダーファイル内に含まれています。構造体については、次の節で説明します。

SASL プラグインに関する重要な構造体

libsasl とプラグイン間の通信は、次の構造体を使って実現されています。

- sasl_utils_t – sasl_utils_t 構造体には、さまざまなユーティリティー関数と 3 つのコンテキスト情報が含まれています。

  この構造体に含まれている各種ユーティリティー関数は、プラグイン開発者向けの簡易関数として機能します。それらの関数の多くは、libsasl の公開インタフェースへのポインタになっています。プラグインは、何らかの理由で SASL コンシューマになる必要がある場合を除き、libsasl を直接呼び出す必要はありません。

  libsasl は sasl_utils_t 用に 3 つのコンテキストを作成します。

  - sasl_conn_t *conn
  - sasl_rand_t *rpool
  - void *getopt_context

  sasl_utils_t 内の conn 変数は、たとえばプラグインの読み込み時のように、実際には特定の接続に関連付けられていない場合があります。それ以外の場合、conn は SASL コンシューマの SASL 接続コンテキストです。rpool 変数は乱数生成関数用です。getopt_context は、getopt() 関数で使用すべきコンテキストです。

  sasl_getopt_t(3SASL)、sasl_log_t(3SASL)、および sasl_getcallback_t(3SASL)
- sasl_out_params_t – libsasl は、sasl_out_params_t 構造体を作成し、それをクライアントまたはサーバーの mech_step() に渡します。この構造体を通じて libsasl に伝えられる情報は、 認証の状態、authid、authzid、maxbuf、折衝済みの ssf、およびデータの符号化/復号化に関する情報です。
- sasl_client_params_t – sasl_client_params_t 構造体は、libsasl がクライアントの状態を特定の SASL クライアント機構に渡すために使用されます。この状態データの送信には、クライアント機構のエントリポイント

mech_new()、mech_step()、および mech_idle() が使用されます。また、canon_user_client() エントリポイントにもクライアント状態を渡す必要があります。

- sasl_server_params_t – sasl_server_params_t 構造体は、サーバー側において、sasl_client_params_t に似た機能を果たします。

クライアントプラグイン

クライアントプラグインは、クライアント側の SASL 折衝を管理するために使用されます。クライアントプラグインは通常、対応するサーバープラグインとともにパッケージ化されます。1 つのクライアントプラグインには、1 つ以上のクライアント側の SASL 機構が含まれています。各 SASL クライアント機構は、認証のサポートに加え、任意で整合性と機密性をサポートします。各機構が自身の機能について提供する情報は、次のとおりです。

- SSF の最大値
- セキュリティーフラグの最大値
- プラグイン機能
- プラグインを使用するためのコールバックとプロンプト ID

クライアントプラグインは sasl_client_plug_init() をエクスポートする必要があります。libsasl は sasl_client_plug_init() を呼び出すことで、プラグインをクライアント用に初期化します。プラグインは sasl_client_plug_t 構造体を返します。sasl_client_plug_t が提供する次のエントリポイントは、libsasl が機構を呼び出す際に使用されます。

- mech_new() – クライアントは接続開始時に sasl_client_start() を呼び出しますが、この関数が mech_new() を使用します。mech_new() は機構に固有の初期化を実行します。必要に応じて、接続コンテキストが割り当てられます。
- mech_step() – mech_step() は、sasl_client_start() と sasl_client_step() から呼び出されます。mech_new() が呼び出されたあと、mech_step() がクライアント側で認証を実行します。認証に成功した場合、mech_step() から SASL_OK が返されます。追加のデータが必要な場合、SASL_CONTINUE が返されます。認証に失敗した場合、SASL エラーコードが返されます。エラーが発生した場合、seterror() が呼び出されます。認証に成功した場合、mech_step() は、関連するセキュリティー層の情報とコールバックを含む sasl_out_params_t 構造体を返す必要があります。canon_user() 関数はこの構造体の一部です。canon_user() は、クライアントが認証 ID と承認 ID を受信した際に呼び出される必要があります。
- mech_dispose() – mech_dispose() は、コンテキストが安全にクローズできる場合に呼び出されます。mech_dispose() は sasl_dispose() によって呼び出されます。
- mech_free() – mech_free() は libsasl の終了時に呼び出されます。mech_free() によって、そのプラグインに対するすべての大域状態が解放されます。

サーバープラグイン

サーバープラグインは、サーバー側の SASL 折衝を管理するために使用されます。サーバープラグインは通常、対応するクライアントプラグインとともにパッケージ化されます。1 つのサーバープラグインには、1 つ以上のサーバー側の SASL 機構が含まれています。各 SASL サーバー機構は、認証のサポートに加え、オプションで整合性と機密性をサポートします。各機構が自身の機能について提供する情報は、次のとおりです。

- SSF の最大値
- セキュリティーフラグの最大値
- プラグイン機能
- プラグインを使用するためのコールバックとプロンプト ID

サーバープラグインは `sasl_server_plug_init()` をエクスポートする必要があります。`libsasl` は `sasl_server_plug_init()` を呼び出すことで、プラグインをサーバー用に初期化します。プラグインは `sasl_server_plug_t` 構造体を返します。`sasl_server_plug_t` が提供する次のエントリポイントは、`libsasl` が機構を呼び出す際に使用されます。

- `mech_new()` – サーバーは接続開始時に `sasl_server_start()` を呼び出しますが、この関数が `mech_new()` を使用します。`mech_new()` は機構に固有の初期化を実行します。`mech_new()` は必要に応じて接続コンテキストを割り当てます。
- `mech_step()` – `mech_step()` は、`sasl_server_start()` と `sasl_server_step()` から呼び出されます。`mech_new()` が呼び出されたあと、`mech_step()` がサーバー側で認証を実行します。認証に成功した場合、`mech_step()` から SASL_OK が返されます。追加のデータが必要な場合、SASL_CONTINUE が返されます。認証に失敗した場合、SASL エラーコードが返されます。エラーが発生した場合、`seterror()` が呼び出されます。認証に成功した場合、`mech_step()` は、関連するセキュリティー層の情報とコールバックを含む `sasl_out_params_t` 構造体を返す必要があります。`canon_user()` 関数はこの構造体の一部です。`canon_user()` は、サーバーが認証 ID と承認 ID を受信した際に呼び出される必要があります。`canon_user()` 関数が呼び出されると、`propctx` が設定されます。認証の標準化前に、必要とされるすべての補助プロパティー要求が実行される必要があります。認証の標準化後に、承認 ID 検索が実行されます。

  `mech_step()` 関数は SASL_OK を返す前に、`sasl_out_params_t` 内のすべての関連フィールドを設定する必要があります。これらのフィールドは次の機能を担っています。

  - `doneflag` – 交換が完了したことを示します
  - `maxoutbuf` – セキュリティー層の最大出力サイズを示します
  - `mech_ssf` – セキュリティー層に対して提供された SSF
  - `encode()` – `sasl_encode()`、`sasl_encodev()`、および `sasl_decode()` によって呼び出されます

  - decode() – sasl_encode()、sasl_encodev()、および sasl_decode() によって呼び出されます
  - encode_context() – sasl_encode()、sasl_encodev()、および sasl_decode() によって呼び出されます
  - decode_context() – sasl_encode()、sasl_encodev()、および sasl_decode() によって呼び出されます
- mech_dispose() – mech_dispose() は、コンテキストが安全にクローズできる場合に呼び出されます。mech_dispose() は sasl_dispose() によって呼び出されます。
- mech_free() – mech_free() は libsasl の終了時に呼び出されます。mech_free() によって、そのプラグインに対するすべての大域状態が解放されます。
- setpass() はユーザーのパスワードを設定します。setpass() を使えば、機構は内部パスワードを持つことができます。
- mech_avail() は sasl_listmech() によって呼び出され、指定されたユーザーが特定の機構を利用できるかどうかを検査します。mech_avail() は新しいコンテキストを作成できるため、mech_new() への呼び出しを回避できます。パフォーマンスに影響が出なければ、この方法でコンテキストを作成します。

ユーザー標準化プラグイン

標準化プラグインは、クライアント側とサーバー側の両方で、認証名と承認名に対する代替標準化サポートを提供します。標準化プラグインを読み込む際には、sasl_canonuser_plug_init() が使用されます。標準化プラグインは次の要件を満たす必要があります。

- 標準化名が出力バッファーにコピーされる必要があります。
- 同じ入力バッファーが出力バッファーとして使用できる必要があります。
- 標準化プラグインは、認証 ID、承認 ID のどちらか一方しか存在しない場合でも、正常に機能する必要があります。

ユーザー標準化プラグインは sasl_canonuser_init() 関数をエクスポートする必要があります。sasl_canonuser_init() 関数は、sasl_canonuser_plug_t を返すことで、必要なエントリポイントを確立する必要があります。ユーザー標準化プラグインは少なくとも、sasl_canonuser_plug_t 構造体のメンバー canon_user_client() または canon_user_server() のどちらか 1 つを実装している必要があります。

補助プロパティー (auxprop) プラグイン

auxprop プラグインは、authid と authzid の両方に対する補助プロパティーを検索するためのサポートを、SASL サーバー側で提供します。たとえば、アプリケーションによっては、内部認証用としてユーザーパスワードを検索したい場合があります。sasl_auxprop_plug_init() 関数は auxprop プラグインを初期化する際に使用され、sasl_auxpropr_plug_t 構造体を返します。

auxprop プラグインを正しく実装するには、sasl_auxprop_plug_t 構造体の auxprop_lookup メンバーを実装する必要があります。auxprop_lookup() 関数はユーザー名の標準化後に、標準化されたユーザー名を指定して呼び出されます。続いてプラグインは、要求された補助プロパティーの取得に必要な任意の検索を実行できます。

注 – Sun Microsystems, Inc. は現在、auxprop プラグインを提供していません。

SASL プラグイン開発のガイドライン

ここでは、SASL プラグイン開発に関して、いくつかの追加の指針を提供します。

SASL プラグインにおけるエラーレポート

適切なエラーレポートは、認証エラーの原因究明時やその他のデバッグ時に役立ちます。プラグイン開発者には、sasl_utils_t 構造体の sasl_seterror() コールバック経由で特定の接続に関する詳細なエラー情報を提供することを強くお勧めします。

SASL プラグインにおけるメモリー割り当て

SASL におけるメモリー割り当ての一般規則は、「開発者自身が割り当てたメモリーは、それらが不要になった時点で漏れなく解放する」というものです。この規則に従えば、パフォーマンスと移植性が改善されるほか、メモリーリークも防止できます。

SASL 折衝順序の設定

プラグイン機構は、クライアントとサーバー間で交わされる SASL 会話の順序を、次のフラグに基づいて設定することができます。

- SASL_FEAT_WANT_CLIENT_FIRST – クライアント側が交換を開始します。
- SASL_FEAT_WANT_SERVER_LAST – サーバーが最後のデータをクライアントに送信します。

どちらのフラグも設定されていない場合、機構プラグインは内部的に順序を設定します。その場合、機構は、送信すべきデータの有無を、クライアントとサーバーの両方で検査する必要があります。なお、「クライアントが最初に送信する」を選択できるのは、プロトコルが初期応答を許可する場合だけです。

「サーバーが最後に送信する」を選択した場合、ステップ関数が SASL_OK を返す際にプラグインが *serverout を設定する必要があります。「サーバーが最後に送信す

る」を決して使用しない機構は、*serverout に NULL を設定する必要があります。「サーバーが最後に送信する」を常に使用する機構は、*serverout に成功データを設定する必要があります。

◆ ◆ ◆ 第 8 章

Oracle Solaris 暗号化フレームワークの紹介

Solaris 暗号化フレームワークは、Solaris オペレーティングシステム上のアプリケーションが暗号化サービスを使用または提供できるようにするためのアーキテクチャーです。このフレームワークとの相互作用はすべて、RSA Security Inc. PKCS#11 Cryptographic Token Interface (Cryptoki) に基づいています。PKCS#11 は、RSA Security Inc. の研究部門である RSA Laboratories によって開発された製品です。この章で説明する Solaris 暗号化フレームワークに関する内容は、次のとおりです。

Oracle Solaris の暗号化に関する用語

暗号化サービスを取得するアプリケーション、ライブラリ、またはカーネルモジュールは「コンシューマ」と呼ばれます。フレームワーク経由でコンシューマに暗号化サービスを提供するアプリケーションは「プロバイダ」または「プラグイン」と呼ばれます。暗号化処理を実装するソフトウェアは「機構」と呼ばれます。機構は単なるアルゴリズムではなく、アルゴリズムの適用方法をも含みます。たとえば、DES アルゴリズムを認証に適用する場合、それは 1 つの独立した機構であるとみなされます。DES をブロック単位の暗号化に適用する場合、それは別の機構になります。

「トークン」とは、暗号化機能を持つデバイスを抽象化した概念です。また、トークンには暗号化処理で使用する情報を格納することもできます。1 つのトークンで 1 つ以上の機構をサポートできます。トークンは、アクセラレータボードの場合と同様に、ハードウェアを表現できます。純粋なソフトウェアを表現するトークンは「ソフトトークン」と呼ばれます。トークンは溝穴 (スロット) に差

し込む (プラグイン) ことができます (これも物理的なたとえです)。スロットは、暗号化サービスを使用するアプリケーションに対する接続ポイントです。

プロバイダ用の特定のスロットのほかに、Solaris 実装は「メタスロット」と呼ばれる特殊なスロットを提供します。メタスロットは、Solaris 暗号化フレームワークライブラリ (`libpkcs11.so`) のコンポーネントです。メタスロットは、フレームワークにインストールされているすべてのトークンとスロットの機能を結合させて単一の仮想スロットで提供するコンポーネントです。メタスロットにより、事実上、アプリケーションから利用可能なすべての暗号化サービスに単一のスロットを通じて透過的に接続できるようになります。アプリケーションが暗号化サービスを要求すると、メタスロットはもっとも適したスロットを示し、これによりスロットの選択処理が簡単になります。それとは異なるスロットが必要となる場合がありますが、その場合はアプリケーションが個別に明示的な検索を実行する必要があります。メタスロットは自動的に有効になり、システム管理者の明示的な操作によってのみ無効にできます。

「セッション」とは、暗号化サービスを使用するアプリケーションとトークン間に確立される接続のことです。PKCS #11 標準で使用されるオブジェクトには、次の 2 種類があります。 トークンオブジェクトとセッションオブジェクトで
す。「セッションオブジェクト」は一時的なオブジェクト、つまりセッション期間内でのみ存在できるオブジェクトです。セッション終了後も存続するオブジェクトが「トークンオブジェクト」と呼ばれます。

トークンオブジェクトのデフォルトの格納場所は、`$HOME/.sunw/pkcs11_softtoken` です。また、トークンオブジェクトを `$SOFTTOKEN_DIR/pkcs11_softtoken` に格納することも可能です。非公開のトークンオブジェクトは個人識別番号 (Personal Identification Number、PIN) によって保護されます。トークンオブジェクトを作成または変更するユーザーは、非公開のトークンオブジェクトにアクセスするのでない限り、必ず認証を受ける必要があります。

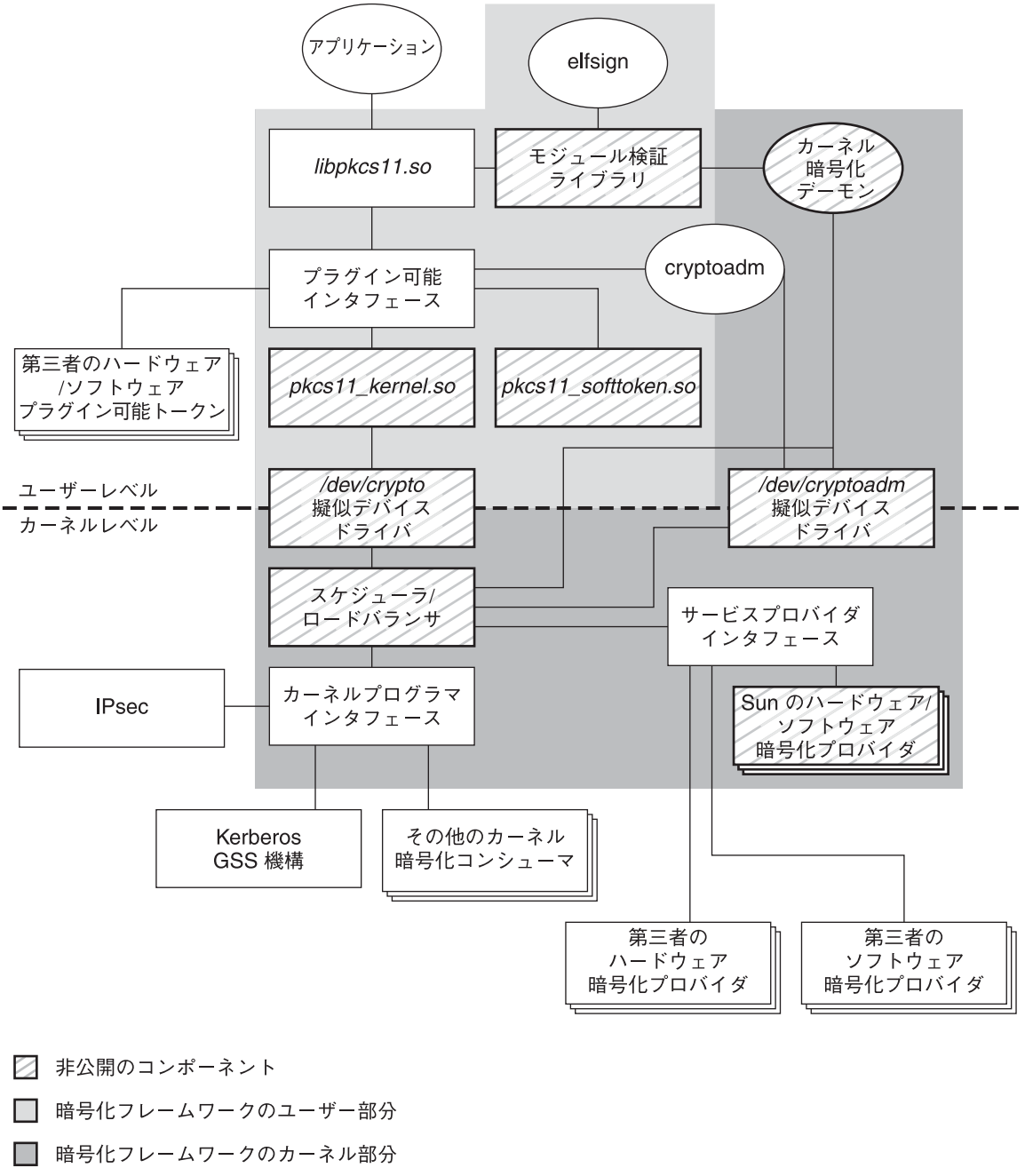
暗号化フレームワークの概要

暗号化フレームワークは Solaris OS の一部であり、Sun Microsystems, Inc. および第三者の開発元による暗号化サービスを提供します。このフレームワークは、次のようなさまざまなサービスを提供します。

- メッセージ暗号化とメッセージダイジェスト
- メッセージ認証コード (Message authentication Code、MAC)
- デジタル署名
- 暗号化サービスにアクセスするためのアプリケーションプログラミングインタフェース (API)
- 暗号化サービスを提供するためのサービスプロバイダインタフェース (SPI)
- 暗号化リソースを管理するための管理コマンド

次の図は、暗号化フレームワークの概要を示したものです。図の淡いグレーの部分は、暗号化フレームワークのユーザーレベル領域を示します。濃いグレーの部分は、フレームワークのカーネルレベル領域を表します。非公開のソフトウェアは、背景に斜線が入っています。

図 8–1　Oracle Solaris 暗号化フレームワークの概要



暗号化フレームワークのコンポーネント

暗号化フレームワークの各コンポーネントの説明は、次のとおりです。

- `libpkcs11.so` – このフレームワークは、RSA Security Inc. PKCS#11 Cryptographic Token Interface (Cryptoki) 経由でのアクセスを提供します。アプリケーションは `libpkcs11.so` ライブラリにリンクする必要があります。このライブラリは RSA PKCS#11 v2.11 標準を実装したものです。
- プラグイン可能インタフェース – プラグイン可能インタフェースは、Sun Microsystems, Inc. および第三書の開発元が提供する PKCS #11 暗号化サービスのためのサービスプロバイダインタフェース (SPI) です。プロバイダはユーザーレベルのライブラリです。プロバイダは、ハードウェアまたはソフトウェアとして使用可能な暗号化サービスを利用して実装されます。
- `pkcs11_softtoken.so` – Sun Microsystems, Inc. が提供する、ユーザーレベルの暗号化機構が含まれた非公開の共有オブジェクト。`pkcs11_softtoken(5)` ライブラリは RSA PKCS#11 v2.11 標準を実装しています。
- `pkcs11_kernel.so` – カーネルレベルの暗号化機構にアクセスするための非公開の共有オブジェクト。`pkcs11_kernel(5)` は RSA PKCS#11 v2.11 仕様を実装しています。 `pkcs11_kernel.so` は、カーネルのサービスプロバイダインタフェースにプラグインされた暗号化サービスに対する PKCS#11 ユーザーインタフェースを提供します。
- `/dev/crypto` 擬似デバイスドライバ – カーネルレベルの暗号化機構を使用するための非公開擬似デバイスドライバ。この情報を提供する目的は、擬似デバイスドライバを誤って削除しないように注意を促すことです。
- スケジューラ/ロードバランサ – 暗号化サービス要求の使用、負荷分散、およびディスパッチを調整する役割を担うカーネルソフトウェア。
- カーネルプログラマインタフェース – 暗号化サービスを使用するカーネルレベルコンシューマ用のインタフェース。IPSec プロトコルと kerberos GSS 機構は典型的な暗号化コンシューマです。

注 – このインタフェースを利用するには、Sun Microsystems, Inc. と連絡を取る必要があります。詳細については、`solaris-crypto-api@sun.com` 宛てに電子メールをお送りください。

- サービスプロバイダインタフェース – カーネルレベルの暗号化サービスプロバイダ用の SPI。これらのサービスは、ハードウェア上に実装することも、ソフトウェア上に実装することもできます。この SPI を使用するプロバイダは、Solaris カーネルから特殊なルーチンをインポートする必要があります。これらのルーチンを使用すれば、モジュールとデバイスドライバはサービスの登録と削除を行えます。また、これらのルーチンは状態の変化をフレームワークに通知します。このフレームワークでは、プロバイダが特定のルーチンをエクスポートする必要も

あります。エクスポートされたルーチンを使用することで、暗号化フレームワークのコンシューマやその他のコンポーネントはプロバイダに要求を送信することができます。

- **Sun HW** 暗号化プロバイダおよび **Sun SW** 暗号化プロバイダ – Sun Microsystems, Inc. が提供するカーネルレベルの暗号化サービス。HW は「ハードウェア」の暗号化サービス (アクセラレータボードなど) を指します。SW は、暗号化アルゴリズムの実装など、暗号化サービスを提供するカーネルモジュールを指します。
- カーネル暗号化フレームワークデーモン – 暗号化処理で使用されるシステムリソースを管理する役割を担う非公開デーモン。また、暗号化プロバイダを検証するのも、このデーモンの役割です。
- モジュール検証ライブラリ – Solaris 暗号化フレームワークがインポートするすべてのバイナリの整合性と認証性を検証するための非公開ライブラリ。
- `elfsign` – 第三者の暗号化サービスプロバイダに提供されるユーティリティー。`elfsign` は、Sun の証明書を要求する場合に使用します。また、プロバイダは `elfsign` を使用することで、Solaris 暗号化フレームワークにプラグインされたバイナリ、つまり elf オブジェクトに実際に署名することもできます。
- `/dev/cryptoadm` 擬似デバイスドライバ – `cryptoadm(1M)` がカーネルレベルの暗号化機構を管理する際に使用する非公開擬似デバイスドライバ。この情報を提供する目的は、擬似デバイスドライバを誤って削除しないように注意を促すことです。
- `cryptoadm` – 管理者が暗号化サービスを管理するためのユーザーレベルのコマンド。 `cryptoadm` で行う一般的な作業は、暗号化プロバイダとその機能の一覧表示です。また、セキュリティーポリシーに従って暗号化機構を無効化および有効化する際も、`cryptoadm` を使用します。

暗号化技術を扱う開発者が知っておくべきこと

ここでは、Solaris 暗号化フレームワークにプラグイン可能な 4 種類のアプリケーションを開発するための要件について説明します。

ユーザーレベルのコンシューマ開発者に対する要件

ユーザーレベルのコンシューマを開発する開発者は、次の点に留意する必要があります。

- `<security/cryptoki.h>` をインクルードします。
- すべての呼び出しを PKCS #11 インタフェース経由で行なってください。
- `libpkcs11.so` をリンクします。
- ライブラリ内で `C_Finalize()` 関数を呼び出すべきではありません。

詳細については、第 9 章「ユーザーレベルの暗号化アプリケーションとプロバイダの記述」を参照してください。

ユーザーレベルのプロバイダ開発者に対する要件

ユーザーレベルのプロバイダを開発する開発者は、次の点に留意する必要があります。

- プロバイダをスタンドアロンとして設計します。プロバイダ共有オブジェクトはアプリケーションのリンク先になるような完全なライブラリでなくてもかまいませんが、必要なシンボルはすべてプロバイダ内に存在している必要があります。プロバイダが `dlopen(3C)` によって RTLD_GROUP モードおよび RTLD_NOW モードでオープンされることを前提にしてください。
- PKCS #11 Cryptoki 実装を共有オブジェクトとして作成します。コンシューマアプリケーションに依存しないですむように、この共有オブジェクトには必要なシンボルが含まれている必要があります。
- データクリーンアップ用の `_fini()` ルーチンを提供することを強くお勧めします。ただし、必須ではありません。この方法を使用すると、アプリケーションまたは共有ライブラリが `libpkcs11` とほかのプロバイダライブラリを同時に読み込む場合に、複数の `C_Finalize()` 呼び出し間で衝突が発生するのを防ぐことができます。164 ページの「ユーザーレベルのプロバイダにおけるデータクリーンアップ衝突の回避」を参照してください。
- Sun Microsystems, Inc. に証明書を申請します。290 ページの「プロバイダに署名するための証明書を要求するには」を参照してください。
- 証明書と `elfsign` を使ってバイナリに署名します。291 ページの「プロバイダに署名するには」を参照してください。
- Sun の規約に従って共有オブジェクトをパッケージ化します。付録 F「暗号化プロバイダのパッケージ化と署名」を参照してください。

カーネルレベルのコンシューマ開発者に対する要件

カーネルレベルのコンシューマを開発する開発者は、次の点に留意する必要があります。

- `<sys/crypto/common.h>` と `<sys/crypto/api.h>` をインクルードします。
- すべての呼び出しをカーネルプログラミングインタフェース経由で行います。

カーネルレベルのプロバイダ開発者に対する要件

カーネルレベルのプロバイダを開発する開発者は、次の点に留意する必要があります。

- <sys/crypto/common.h> と <sys/crypto/api.h> をインクルードします。
- 登録、登録解除、および状態提供に必要なルーチンをインポートします。
- カーネル暗号化フレームワークに対してエントリポイントを提供するために、必要なルーチンをエクスポートします。
- サポートされているアルゴリズムの説明を含むデータ構造をエクスポートします。
- 読み込み可能なカーネルモジュールを作成します。
- Sun Microsystems, Inc. に証明書を申請します。290 ページの「プロバイダに署名するための証明書を要求するには」を参照してください。
- 証明書と elfsign を使ってバイナリに署名します。291 ページの「プロバイダに署名するには」を参照してください。
- Sun の規約に従ってカーネルモジュールをパッケージ化します。付録 F 「暗号化プロバイダのパッケージ化と署名」を参照してください。

ユーザーレベルのプロバイダにおけるデータクリーンアップ衝突の回避

暗号化フレームワークにプラグインするユーザーレベルライブラリでは、_fini() 関数を提供する必要があります。_fini() 関数は、ライブラリの読み込み解除時にローダによって呼び出されます。_fini() 関数は、すべてのクリーンアップが正しいタイミングで正しく実行されるために必要です。libpkcs11 を使用するライブラリが C_Finalize() を呼び出すことは想定されていません。なぜなら、libpkcs11 はアプリケーションによって使用されている可能性がある共有ライブラリだからです。

_fini() 関数を提供するには、再配置可能オブジェクトのプログラムデータセクション内に .fini セクションを作成する必要があります。.fini セクションは実行時終了コードブロックを提供します。『リンカーとライブラリ』を参照してください。次のコード例は .fini セクションの設計方法を示しています。

例 8–1　PKCS #11 ライブラリへの _fini() の提供

```
#pragma fini(pkcs11_fini)
static void pkcs11_fini();

/* [... (other library code omitted)] */

static void
pkcs11_fini()
```

例 8–1　PKCS #11 ライブラリへの _fini() の提供　　(続き)

```
{
        (void) pthread_mutex_lock(&pkcs11mutex);

        /* If CRYPTOKI is not initialized, do not clean up */
        if (!initialized) {
                (void) pthread_mutex_unlock(&pkcs11mutex);
                return;
        }

        (void) finalize_routine(NULL_PTR);

        (void) pthread_mutex_unlock(&pkcs11mutex);
}
```

◆ ◆ ◆ 第 9 章

ユーザーレベルの暗号化アプリケーションとプロバイダの記述

この章では、暗号化用の PKCS #11 関数を使用したユーザーレベルのアプリケーションとプロバイダを開発する方法について説明します。次の項目について説明します。

暗号化フレームワークについての詳細は、第 8 章「Oracle Solaris 暗号化フレームワークの紹介」を参照してください。

cryptoki ライブラリの概要

Solaris 暗号化フレームワーク内のユーザーレベルアプリケーションは、`libpkcs11.so` モジュールで提供される `cryptoki` ライブラリ経由で PKCS #11 関数にアクセスします。`pkcs11_softtoken.so` モジュールは Sun Microsystems, Inc. が提供する PKCS #11 ソフトトークン実装であり、さまざまな暗号化機構を提供します。このソフトトークンプラグインが機構のデフォルトソースになります。暗号化機構の提供は、Sun 以外のプラグインを通じて行うことも可能です。

ここでは、このソフトトークンによってサポートされている PKCS #11 関数と戻り値のリストを示します。戻りコードは、フレームワークにプラグインされるプロバイダごとに異なります。また、いくつかの一般的な関数についても説明します。`cryptoki` ライブラリに含まれるすべての要素の完全な説明が必要な場合は、`libpkcs11(3LIB)` のマニュアルページを参照するか、RSA Laboratories Web サイトの「PKCS #11: Cryptographic Token Interface Standard」を参照してください。

PKCS #11 関数リスト

次のリストは、Solaris 暗号化フレームワーク内の pkcs11_softtoken.so がサポートする PKCS #11 関数をカテゴリ別に示したものです。

- 一般用途 – C_Initialize()、C_Finalize()、C_GetInfo()、C_GetFunctionList()
- セッション管理 – C_OpenSession()、C_CloseSession()、C_GetSessionInfo()、C_CloseAllSessions()、C_Login()、
- スロット管理およびトークン管理 – C_GetSlotList()、C_GetSlotInfo()、C_GetMechanismList()、C_GetMechanismInfo()、C_SetPIN(
- 暗号化と復号化 – C_EncryptInit()、C_Encrypt()、C_EncryptUpdate()、C_EncryptFinal()、C_DecryptInit()、C_D C_DecryptUpdate()、C_DecryptFinal()
- メッセージダイジェスト – C_DigestInit()、C_Digest()、C_DigestKey()、C_DigestUpdate()、C_DigestFinal()
- 署名と MAC 適用 – C_Sign()、C_SignInit()、C_SignUpdate()、C_SignFinal()、C_SignRecoverInit()、C_SignRecov
- 署名検証 – C_Verify()、C_VerifyInit()、C_VerifyUpdate()、C_VerifyFinal()、C_VerifyRecoverInit()、C
- 二重目的の暗号化関数 – C_DigestEncryptUpdate()、C_DecryptDigestUpdate()、C_SignEncryptUpdate()、C_DecryptVerif
- 乱数生成 – C_SeedRandom()、C_GenerateRandom()
- オブジェクト管理 – C_CreateObject()、C_DestroyObject()、C_CopyObject()、C_FindObjects()、C_FindObjectsInit C_GetAttributeValue()、C_SetAttributeValue()
- 鍵管理 – C_GenerateKey()、C_GenerateKeyPair()、C_DeriveKey()

PKCS #11 を使用するための関数

ここでは、PKCS #11 を使用するための関数のうち、次のものについて説明します。

- 169 ページの「PKCS #11 関数: C_Initialize()」
- 169 ページの「PKCS #11 関数: C_GetInfo()」
- 170 ページの「PKCS #11 関数: C_GetSlotList()」
- 170 ページの「PKCS #11 関数: C_GetTokenInfo()」
- 171 ページの「PKCS #11 関数: C_OpenSession()」
- 172 ページの「PKCS #11 関数: C_GetMechanismList()」

注 – libpkcs11.so ライブラリではすべての PKCS #11 関数が利用可能になっています。C_GetFunctionList() 関数を使用して利用可能な関数のリストを取得する必要はありません。

PKCS #11 関数: C_Initialize()

C_Initialize() は PKCS #11 ライブラリを初期化します。C_Initialize() の構文は次のとおりです。

```
C_Initialize(CK_VOID_PTR pInitArgs);
```

pInitArgs は、空値 NULL_PTR または CK_C_INITIALIZE_ARGS 構造体へのポインタです。NULL_PTR が指定された場合、ライブラリは Solaris の相互排他ロックをロックプリミティブとして使用することで、複数スレッドによる内部共有構造体へのアクセスを制御します。Solaris 暗号化フレームワークは相互排他ロックを受け入れないことに注意してください。この実装版の cryptoki ライブラリは、マルチスレッド処理を安全かつ効率的に処理しますので、NULL_PTR を使用することをお勧めします。またアプリケーション内で、*pInitArgs* を使用して CKF_LIBRARY_CANT_CREATE_OS_THREADS などのフラグを設定することも可能です。C_Finalize() は、アプリケーションが PKCS #11 ライブラリを使用し終わったことを通知します。

注 – C_Finalize() は決してライブラリ内から呼び出してはいけません。一般に、C_Finalize() を呼び出してセッションをクローズすることはアプリケーションの責任です。

CKR_FUNCTION_FAILED、CKR_GENERAL_ERROR、CKR_HOST_MEMORY、CKR_OK に加え、C_Initialize() は次の戻り値を使用します。

- CKR_ARGUMENTS_BAD
- CKR_CANT_LOCK
- CKR_CRYPTOKI_ALREADY_INITIALIZED – このエラーは致命的ではありません。

PKCS #11 関数: C_GetInfo()

C_GetInfo() は、cryptoki ライブラリに関する開発元情報とバージョン情報を取得します。C_GetInfo() の構文は次のとおりです。

```
C_GetInfo(CK_INFO_PTR pInfo);
```

C_GetInfo() は次の値を返します。

- cryptokiVersion = 2, 11
- manufacturerID = Sun Microsystems, Inc.

CKR_FUNCTION_FAILED、CKR_GENERAL_ERROR、CKR_HOST_MEMORY、CKR_OK に加え、C_GetInfo() は次の戻り値を使用します。

- CKR_ARGUMENTS_BAD
- CKR_CRYPTOKI_NOT_INITIALIZED

PKCS #11 関数: C_GetSlotList()

C_GetSlotList() は、利用可能なスロットのリストを取得します。pkcs11_softtoken.so 以外の暗号化プロバイダがインストールされていない場合、C_GetSlotList() から返されるのは、デフォルトのスロットだけです。C_GetSlotList() の構文は次のとおりです。

```
C_GetSlotList(CK_BBOOL tokenPresent, CK_SLOT_ID_PTR pSlotList,
CK_ULONG_PTR pulCount);
```

tokenPresent に TRUE を設定した場合、トークンが存在するスロットだけに検索対象が限定されます。

pSlotList に NULL_PTR を設定した場合、C_GetSlotlist() はスロット数だけを返します。*pulCount* はスロット数を格納する場所へのポインタです。

pSlotList にスロットを格納するバッファーへのポインタを設定した場合、**pulCount* には CK_SLOT_ID 要素の予想最大数を設定します。戻り時には、**pulCount* に CK_SLOT_ID 要素の実際の個数が設定されます。

通常、PKCS #11 アプリケーションは C_GetSlotList() を 2 回呼び出します。1 回目の C_GetSlotList() 呼び出しでは、メモリーを割り当てる目的でスロット数を取得します。そして 2 回目の C_GetSlotList() 呼び出しでは、スロットを取得します。

注 – スロットの順序は保証されません。スロットの順序は、PKCS #11 ライブラリを読み込むたびに変わる可能性があります。

CKR_FUNCTION_FAILED、CKR_GENERAL_ERROR、CKR_HOST_MEMORY、CKR_OK に加え、C_GetSlotlist() は次の戻り値を使用します。

- CKR_ARGUMENTS_BAD
- CKR_BUFFER_TOO_SMALL
- CKR_CRYPTOKI_NOT_INITIALIZED

PKCS #11 関数: C_GetTokenInfo()

C_GetTokenInfo() は、特定のトークンに関する情報を取得します。C_GetTokenInfo() の構文は次のとおりです。

```
C_GetTokenInfo(CK_SLOT_ID slotID, CK_TOKEN_INFO_PTR pInfo);
```

slotID は、目的のトークンに対するスロットの ID です。*slotID* は、`C_GetSlotList()` から返された有効な ID でなければなりません。*pInfo* は、トークン情報を格納する場所へのポインタです。

`pkcs11_softtoken.so` がインストールされている唯一のプロバイダである場合、`C_GetTokenInfo()` は次のフィールドと値を返します。

- ラベル – Sun Software PKCS#11 softtoken。
- フラグ – `CKF_DUAL_CRYPTO_OPERATIONS`、`CKF_TOKEN_INITIALIZED`、`CKF_RNG`、`CKF_USER_PIN_INITIALIZ` および `CKF_LOGIN_REQUIRED`。これらのフラグには 1 が設定されます。
- ulMaxSessionCount – `CK_EFFECTIVELY_INFINITE` が設定されます。
- ulMaxRwSessionCount – `CK_EFFECTIVELY_INFINITE` が設定されます。
- ulMaxPinLen – 256 が設定されます。
- ulMinPinLen – 1 が設定されます。
- ulTotalPublicMemory には `CK_UNAVAILABLE_INFORMATION` が設定されます。
- ulFreePublicMemory には `CK_UNAVAILABLE_INFORMATION` が設定されます。
- ulTotalPrivateMemory には `CK_UNAVAILABLE_INFORMATION` が設定されます。
- ulFreePrivateMemory には `CK_UNAVAILABLE_INFORMATION` が設定されます。

CKR_FUNCTION_FAILED、CKR_GENERAL_ERROR、CKR_HOST_MEMORY、CKR_OK に加え、`C_GetSlotlist()` は次の戻り値を使用します。

- `CKR_ARGUMENTS_BAD`
- `CKR_BUFFER_TOO_SMALL`
- `CKR_CRYPTOKI_NOT_INITIALIZED`
- `CKR_SLOT_ID_INVALID`

次の戻り値は、プラグインされたハードウェアトークンに関するものです。

- `CKR_DEVICE_ERROR`
- `CKR_DEVICE_MEMORY`
- `CKR_DEVICE_REMOVED`
- `CKR_TOKEN_NOT_PRESENT`
- `CKR_TOKEN_NOT_RECOGNIZED`

PKCS #11 関数: `C_OpenSession()`

`C_OpenSession()` を使えば、アプリケーションは特定のスロット内の特定のトークンとの間で暗号化セッションを開始できます。`C_OpenSession()` の構文は次のとおりです。

```
C_OpenSession(CK_SLOT_ID slotID, CK_FLAGS flags, CK_VOID_PTR pApplication,
CK_NOTIFY Notify, CK_SESSION_HANDLE_PTR phSession);
```

slotID はスロットの ID です。*flags* は、セッションを読み書き可能にするか、あるいは読み取り専用にするかを示します。*pApplication* は、アプリケーションによってコールバック用として定義されたポインタです。*Notify* には、オプションのコールバック関数のアドレスが格納されます。*phSession* は、セッションハンドルの格納場所へのポインタです。

CKR_FUNCTION_FAILED、CKR_GENERAL_ERROR、CKR_HOST_MEMORY、CKR_OK に加え、`C_OpenSession()` は次の戻り値を使用します。

- `CKR_ARGUMENTS_BAD`
- `CKR_CRYPTOKI_NOT_INITIALIZED`
- `CKR_SLOT_ID_INVALID`
- `CKR_TOKEN_WRITE_PROTECTED` – 読み取り専用のトークンで戻されます。

次の戻り値は、プラグインされたハードウェアトークンに関するものです。

- `CKR_DEVICE_ERROR`
- `CKR_DEVICE_MEMORY`
- `CKR_DEVICE_REMOVED`
- `CKR_SESSION_COUNT`
- `CKR_SESSION_PARALLEL_NOT_SUPPORTED`
- `CKR_SESSION_READ_WRITE_SO_EXISTS`
- `CKR_TOKEN_NOT_PRESENT`
- `CKR_TOKEN_NOT_RECOGNIZED`

PKCS #11 関数: `C_GetMechanismList()`

`C_GetMechanismList()` は、指定されたトークンがサポートする機構タイプのリストを取得します。`C_GetMechanismList()` の構文は次のとおりです。

```
C_GetMechanismList(CK_SLOT_ID slotID, CK_MECHANISM_TYPE_PTR pMechanismList,
CK_ULONG_PTR pulCount);
```

slotID は、目的のトークンに対するスロットの ID です。*pulCount* は機構数を格納する場所へのポインタです。*pMechanismList* に NULL_PTR を設定した場合、**pulCount* に機構数が返されます。それ以外の場合、**pulCount* にはリストのサイズを、*pMechanismList* にはリストを格納するバッファーへのポインタを、それぞれ設定する必要があります。

PKCS #11 ソフトトークンがプラグインされている場合、`C_GetMechanismList()` から返されるサポート機構リストは、次のとおりです。

- `CKM_AES_CBC`
- `CKM_AES_CBC_PAD`
- `CKM_AES_ECB`
- `CKM_AES_KEY_GEN`
- `CKM_DES_CBC`
- `CKM_DES_CBC_PAD`

- CKM_DES_ECB
- CKM_DES_KEY_GEN
- CKM_DES_MAC
- CKM_DES_MAC_GENERAL
- CKM_DES3_CBC
- CKM_DES3_CBC_PAD
- CKM_DES3_ECB
- CKM_DES3_KEY_GEN
- CKM_DH_PKCS_DERIVE
- CKM_DH_PKCS_KEY_PAIR_GEN
- CKM_DSA
- CKM_DSA_KEY_PAIR_GEN
- CKM_DSA_SHA_1
- CKM_MD5
- CKM_MD5_KEY_DERIVATION
- CKM_MD5_RSA_PKCS
- CKM_MD5_HMAC
- CKM_MD5_HMAC_GENERAL
- CKM_PBE_SHA1_RC4_128
- CKM_PKCS5_PBKD2
- CKM_RC4
- CKM_RC4_KEY_GEN
- CKM_RSA_PKCS
- CKM_RSA_X_509
- CKM_RSA_PKCS_KEY_PAIR_GEN
- CKM_SHA_1
- CKM_SHA_1_HMAC_GENERAL
- CKM_SHA_1_HMAC
- CKM_SHA_1_KEY_DERIVATION
- CKM_SHA_1_RSA_PKCS
- CKM_SSL3_KEY_AND_MAC_DERIVE
- CKM_SSL3_MASTER_KEY_DERIVE
- CKM_SSL3_MASTER_KEY_DERIVE_DH
- CKM_SSL3_MD5_MAC
- CKM_SSL3_PRE_MASTER_KEY_GEN
- CKM_SSL3_SHA1_MAC
- CKM_TLS_KEY_AND_MAC_DERIVE
- CKM_TLS_MASTER_KEY_DERIVE
- CKM_TLS_MASTER_KEY_DERIVE_DH
- CKM_TLS_PRE_MASTER_KEY_GEN

CKR_FUNCTION_FAILED、CKR_GENERAL_ERROR、CKR_HOST_MEMORY、CKR_OK に加え、`C_GetSlotlist()` は次の戻り値を使用します。

- CKR_ARGUMENTS_BAD

- CKR_BUFFER_TOO_SMALL
- CKR_CRYPTOKI_NOT_INITIALIZED
- CKR_SLOT_ID_INVALID

次の戻り値は、プラグインされたハードウェアトークンに関するものです。

- CKR_DEVICE_ERROR
- CKR_DEVICE_MEMORY
- CKR_DEVICE_REMOVED
- CKR_TOKEN_NOT_PRESENT
- CKR_TOKEN_NOT_RECOGNIZED

拡張 PKCS #11 関数

Solaris 暗号化フレームワークでは、標準の PKCS #11 関数のほかに、次の 2 つの簡易関数が提供されています。

- 174 ページの「拡張 PKCS #11 関数: SUNW_C_GetMechSession()」
- 174 ページの「拡張 PKCS #11 関数: SUNW_C_KeyToObject」

拡張 PKCS #11 関数: SUNW_C_GetMechSession()

SUNW_C_GetMechSession() は、簡易関数です。この関数はまず、Solaris 暗号化フレームワークを初期化します。続いてこの関数は、指定された機構との間でセッションを開始します。SUNW_C_GetMechSession() の構文は次のとおりです。

```
SUNW_C_GetMechSession(CK_MECHANISM_TYPE mech, C\
K_SESSION_HANDLE_PTR hSession)
```

mech パラメータでは、使用する機構を指定します。*hSession* は、セッションの格納場所へのポインタです。

SUNW_C_GetMechSession() は内部的に、C_Initialize() を呼び出して cryptoki ライブラリを初期化します。続いて SUNW_C_GetMechSession() は、C_GetSlotList() と C_GetMechanismInfo() を使って利用可能なスロットを検索し、指定された機構を備えたトークンを見つけ出します。目的の機構が見つかった場合、SUNW_C_GetMechSession() は C_OpenSession() を呼び出してセッションをオープンします。

SUNW_C_GetMechSession() は何度も呼び出す必要はありません。しかし、仮に SUNW_C_GetMechSession() を複数回呼び出したとしても、特に問題は生じません。

拡張 PKCS #11 関数: SUNW_C_KeyToObject

SUNW_C_KeyToObject() は、秘密鍵オブジェクトを作成します。呼び出し元のプログラムは、使用すべき機構と未処理の鍵データを指定する必要があります。SUNW_C_KeyToObject() は内部的に、指定された機構に対する鍵の種類を決定しま

す。C_CreateObject() によって汎用的な鍵オブジェクトが作成されます。続いて SUNW_C_KeyToObject() は、 C_GetSessionInfo() と C_GetMechanismInfo() を呼び出してスロットと機構の情報を取得します。そして、C_SetAttributeValue() によって、鍵オブジェクトの属性フラグが機構の種類に従って設定されます。

ユーザーレベルの暗号化アプリケーションの例

この節では、次の例について説明します。

メッセージダイジェストの例

この例では、PKCS #11 関数を使って入力ファイルからダイジェストを作成します。この例では、次の手順を実行します。

1. ダイジェストの機構を指定します。

   この例では、CKM_MD5 ダイジェスト機構が使用されます。

2. 指定されたダイジェストアルゴリズムをサポートするスロットを検索します。

   この例では、Sun の簡易関数 SUNW_C_GetMechSession() を使用しています。SUNW_C_GetMechSession() は、cryptoki ライブラリをオープンします。このライブラリには、Solaris 暗号化フレームワークで使用されるすべての PKCS #11 関数が含まれています。続いて SUNW_C_GetMechSession() は、目的の機構を備えたスロットを検索します。そして、セッションが開始されます。この簡易関数は事実上、C_Initialize() 呼び出し、C_OpenSession() 呼び出し、および指定された機構をサポートするスロットの検索に必要なすべてのコードと同等の機能を備えています。

3. cryptoki の情報を取得します。

   この部分は実際には、メッセージダイジェストの作成に直接関係しませんが、C_GetInfo() 関数の使用法を示す意味でここに含めてあります。この例では開発元の ID を取得しています。その他の情報のオプションとして、バージョンとライブラリデータも取得できます。

4. スロットを使ってダイジェスト処理を実行します。

   この作業におけるメッセージダイジェストの作成手順は次のとおりです。

   a. 入力ファイルをオープンします。

   b. C_DigestInit() を呼び出してダイジェスト処理を初期化します。

   c. C_DigestUpdate() を使ってデータを一度に一部分ずつ処理していきます。

d. ダイジェスト処理を終了するために、C_DigestFinal() を使って完全なダイジェストを取得します。

5. セッションを終了します。

   このプログラムは、C_CloseSession() を使ってセッションをクローズし、C_Finalize() を使ってライブラリをクローズします。

メッセージダイジェストのソースコード例を、次に示します。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

例 9–1 PKCS #11 関数によるメッセージダイジェストの作成

```
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <security/cryptoki.h>
#include <security/pkcs11.h>

#define BUFFERSIZ    8192
#define MAXDIGEST    64

/* Calculate the digest of a user supplied file. */
void
main(int argc, char **argv)
{
    CK_BYTE digest[MAXDIGEST];
    CK_INFO info;
    CK_MECHANISM mechanism;
    CK_SESSION_HANDLE hSession;
    CK_SESSION_INFO Info;
    CK_ULONG ulDatalen = BUFFERSIZ;
    CK_ULONG ulDigestLen = MAXDIGEST;
    CK_RV rv;
    CK_SLOT_ID SlotID;

    int i, bytes_read = 0;
    char inbuf[BUFFERSIZ];
    FILE *fs;
    int error = 0;

    /* Specify the CKM_MD5 digest mechanism as the target */
    mechanism.mechanism = CKM_MD5;
    mechanism.pParameter = NULL_PTR;
    mechanism.ulParameterLen = 0;

    /* Use SUNW convenience function to initialize the cryptoki
     * library, and open a session with a slot that supports
     * the mechanism we plan on using. */
    rv = SUNW_C_GetMechSession(mechanism.mechanism, &hSession);
    if (rv != CKR_OK) {
```

例 9–1 PKCS #11 関数によるメッセージダイジェストの作成 *(続き)*

```
        fprintf(stderr, "SUNW_C_GetMechSession: rv = 0x%.8X\n", rv);
        exit(1);
    }

    /* Get cryptoki information, the manufacturer ID */
    rv = C_GetInfo(&info);
    if (rv != CKR_OK) {
        fprintf(stderr, "WARNING: C_GetInfo: rv = 0x%.8X\n", rv);
    }
    fprintf(stdout, "Manufacturer ID = %s\n", info.manufacturerID);

    /* Open the input file */
    if ((fs = fopen(argv[1], "r")) == NULL) {
        perror("fopen");
        fprintf(stderr, "\n\tusage: %s filename>\n", argv[0]);
        error = 1;
        goto exit_session;
    }

    /* Initialize the digest session */
    if ((rv = C_DigestInit(hSession, &mechanism)) != CKR_OK) {
        fprintf(stderr, "C_DigestInit: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_digest;
    }

    /* Read in the data and create digest of this portion */
    while (!feof(fs) && (ulDatalen = fread(inbuf, 1, BUFFERSIZ, fs)) > 0) {
        if ((rv = C_DigestUpdate(hSession, (CK_BYTE_PTR)inbuf,
                    ulDatalen)) != CKR_OK) {
            fprintf(stderr, "C_DigestUpdate: rv = 0x%.8X\n", rv);
            error = 1;
            goto exit_digest;
        }
        bytes_read += ulDatalen;
    }
    fprintf(stdout, "%d bytes read and digested!!!\n\n", bytes_read);

    /* Get complete digest */
    ulDigestLen = sizeof (digest);
    if ((rv = C_DigestFinal(hSession, (CK_BYTE_PTR)digest,
                &ulDigestLen)) != CKR_OK) {
        fprintf(stderr, "C_DigestFinal: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_digest;
    }

    /* Print the results */
    fprintf(stdout, "The value of the digest is: ");
    for (i = 0; i < ulDigestLen; i++) {
        fprintf(stdout, "%.2x", digest[i]);
    }
    fprintf(stdout, "\nDone!!!\n");

exit_digest:
    fclose(fs);
```

例 9–1 PKCS #11 関数によるメッセージダイジェストの作成 (続き)

```
exit_session:
    (void) C_CloseSession(hSession);

exit_program:
    (void) C_Finalize(NULL_PTR);

    exit(error);

}
```

対称暗号化の例

例 9–2 は、DES アルゴリズムの CBC モードを使って暗号化するための鍵オブジェクトを作成します。このソースコードは次の手順を実行します。

1. 鍵データを宣言します。

   DES と初期化ベクトルを定義します。ここでは初期化ベクトルを静的に宣言していますが、これはあくまでもデモ用です。初期化ベクトルは常に動的に定義し、決して再利用しないでください。

2. 鍵オブジェクトを定義します。

   この作業では、鍵のテンプレートを設定する必要があります。

3. 指定された暗号化機構をサポートするスロットを検索します。

   この例では、Sun の簡易関数 `SUNW_C_GetMechSession()` を使用しています。`SUNW_C_GetMechSession()` は、`cryptoki` ライブラリをオープンします。このライブラリには、Solaris 暗号化フレームワークで使用されるすべての PKCS #11 関数が含まれています。続いて `SUNW_C_GetMechSession()` は、目的の機構を備えたスロットを検索します。そして、セッションが開始されます。この簡易関数は事実上、`C_Initialize()` 呼び出し、`C_OpenSession()` 呼び出し、および指定された機構をサポートするスロットの検索に必要なすべてのコードと同等の機能を備えています。

4. このスロットで暗号化処理を実行します。

   この作業における暗号化の実行手順は次のとおりです。

   a. 入力ファイルをオープンします。

   b. 鍵のオブジェクトハンドルを作成します。

   c. mechanism 構造体を使って暗号化機構を CKM_DES_CBC_PAD に設定します。

   d. `C_EncryptInit()` を呼び出して暗号化処理を初期化します。

   e. `C_EncryptUpdate()` を使ってデータを一度に一部分ずつ処理していきます。

   f. 暗号化処理を終了するために、`C_EncryptFinal()` を使って暗号化データの最後の部分を取得します。

5. このスロットで復号化処理を実行します。

   この作業における復号化の実行手順は次のとおりです。この復号化処理は、テスト目的でのみ提供されています。

   a. `C_DecryptInit()` を呼び出して復号化処理を初期化します。
   b. `C_Decrypt()` で文字列全体を処理します。

6. セッションを終了します。

   このプログラムは、`C_CloseSession()` を使ってセッションをクローズし、`C_Finalize()` を使ってライブラリをクローズします。

対称暗号化のソースコード例を、次に示します。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

例 9–2　PKCS #11 関数による暗号化鍵オブジェクトの作成

```
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <security/cryptoki.h>
#include <security/pkcs11.h>

#define BUFFERSIZ    8192

/* Declare values for the key materials. DO NOT declare initialization
 * vectors statically like this in real life!! */
uchar_t des_key[] = { 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef};
uchar_t des_cbc_iv[] = { 0x12, 0x34, 0x56, 0x78, 0x90, 0xab, 0xcd, 0xef};

/* Key template related definitions. */
static CK_BBOOL truevalue = TRUE;
static CK_BBOOL falsevalue = FALSE;
static CK_OBJECT_CLASS class = CKO_SECRET_KEY;
static CK_KEY_TYPE keyType = CKK_DES;


/* Example encrypts and decrypts a file provided by the user. */
void
main(int argc, char **argv)
{
    CK_RV rv;
    CK_MECHANISM mechanism;
    CK_OBJECT_HANDLE hKey;
    CK_SESSION_HANDLE hSession;
    CK_ULONG ciphertext_len = 64, lastpart_len = 64;
    long ciphertext_space = BUFFERSIZ;
    CK_ULONG decrypttext_len;
    CK_ULONG total_encrypted = 0;
    CK_ULONG ulDatalen = BUFFERSIZ;
```

例 9–2 PKCS#11 関数による暗号化鍵オブジェクトの作成 (続き)

```
int i, bytes_read = 0;
int error = 0;
char inbuf[BUFFERSIZ];
FILE *fs;
uchar_t ciphertext[BUFFERSIZ], *pciphertext, decrypttext[BUFFERSIZ];

/* Set the key object */
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof (class) },
    {CKA_KEY_TYPE, &keyType, sizeof (keyType) },
    {CKA_TOKEN, &falsevalue, sizeof (falsevalue) },
    {CKA_ENCRYPT, &truevalue, sizeof (truevalue) },
    {CKA_VALUE, &des_key, sizeof (des_key) }
};

/* Set the encryption mechanism to CKM_DES_CBC_PAD */
mechanism.mechanism = CKM_DES_CBC_PAD;
mechanism.pParameter = des_cbc_iv;
mechanism.ulParameterLen = 8;

/* Use SUNW convenience function to initialize the cryptoki
 * library, and open a session with a slot that supports
 * the mechanism we plan on using. */
rv = SUNW_C_GetMechSession(mechanism.mechanism, &hSession);

if (rv != CKR_OK) {
    fprintf(stderr, "SUNW_C_GetMechSession: rv = 0x%.8X\n", rv);
    exit(1);
}

/* Open the input file */
if ((fs = fopen(argv[1], "r")) == NULL) {
    perror("fopen");
    fprintf(stderr, "\n\tusage: %s filename>\n", argv[0]);
    error = 1;
    goto exit_session;
}

/* Create an object handle for the key */
rv = C_CreateObject(hSession, template,
    sizeof (template) / sizeof (CK_ATTRIBUTE),
    &hKey);

if (rv != CKR_OK) {
    fprintf(stderr, "C_CreateObject: rv = 0x%.8X\n", rv);
    error = 1;
    goto exit_session;
}


/* Initialize the encryption operation in the session */
rv = C_EncryptInit(hSession, &mechanism, hKey);

if (rv != CKR_OK) {
    fprintf(stderr, "C_EncryptInit: rv = 0x%.8X\n", rv);
```

例 9–2 PKCS #11 関数による暗号化鍵オブジェクトの作成 (続き)

```
        error = 1;
        goto exit_session;
    }

    /* Read in the data and encrypt this portion */
    pciphertext = &ciphertext[0];
    while (!feof(fs) && (ciphertext_space > 0) &&
        (ulDatalen = fread(inbuf, 1, ciphertext_space, fs)) > 0) {
        ciphertext_len = ciphertext_space;

        /* C_EncryptUpdate is only being sent one byte at a
         * time, so we are not checking for CKR_BUFFER_TOO_SMALL.
         * Also, we are checking to make sure we do not go
         * over the alloted buffer size.  A more robust program
         * could incorporate realloc to enlarge the buffer
         * dynamically.     */
        rv = C_EncryptUpdate(hSession, (CK_BYTE_PTR)inbuf, ulDatalen,
            pciphertext, &ciphertext_len);
        if (rv != CKR_OK) {
            fprintf(stderr, "C_EncryptUpdate: rv = 0x%.8X\n", rv);
            error = 1;
            goto exit_encrypt;
        }
        pciphertext += ciphertext_len;
        total_encrypted += ciphertext_len;
        ciphertext_space -= ciphertext_len;
        bytes_read += ulDatalen;
    }

    if (!feof(fs) || (ciphertext_space < 0)) {
        fprintf(stderr, "Insufficient space for encrypting the file\n");
        error = 1;
        goto exit_encrypt;
    }

    /* Get the last portion of the encrypted data */
    lastpart_len = ciphertext_space;
    rv = C_EncryptFinal(hSession, pciphertext, &lastpart_len);
    if (rv != CKR_OK) {
        fprintf(stderr, "C_EncryptFinal: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_encrypt;
    }
    total_encrypted += lastpart_len;

    fprintf(stdout, "%d bytes read and encrypted. Size of the "
        "ciphertext: %d!\n\n", bytes_read, total_encrypted);

    /* Print the encryption results */
    fprintf(stdout, "The value of the encryption is:\n");
    for (i = 0; i < ciphertext_len; i++) {
        if (ciphertext[i] < 16)
            fprintf(stdout, "0%x", ciphertext[i]);
        else
            fprintf(stdout, "%2x", ciphertext[i]);
    }
```

例 9–2　PKCS#11 関数による暗号化鍵オブジェクトの作成　　(続き)

```
    /* Initialize the decryption operation in the session */
    rv = C_DecryptInit(hSession, &mechanism, hKey);

    /* Decrypt the entire ciphertext string */
    decrypttext_len = sizeof (decrypttext);
    rv = C_Decrypt(hSession, (CK_BYTE_PTR)ciphertext, total_encrypted,
        decrypttext, &decrypttext_len);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_Decrypt: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_encrypt;
    }

    fprintf(stdout, "\n\n%d bytes decrypted!!!\n\n", decrypttext_len);

    /* Print the decryption results */
    fprintf(stdout, "The value of the decryption is:\n%s", decrypttext);

    fprintf(stdout, "\nDone!!!\n");

exit_encrypt:
    fclose(fs);

exit_session:
    (void) C_CloseSession(hSession);

exit_program:
    (void) C_Finalize(NULL_PTR);
    exit(error);
}
```

署名と検証の例

ここで説明する例は、RSA 鍵ペアを生成します。そのあと、その鍵ペアで単純な文字列を署名および検証します。この例では、次の手順を実行します。

1. 鍵オブジェクトを定義します。
2. 公開鍵のテンプレートを設定します。
3. 非公開鍵のテンプレートを設定します。
4. メッセージ例を作成します。
5. 鍵ペアを生成する `genmech` 機構を指定します。
6. 鍵ペアに署名する `smech` 機構を指定します。
7. `cryptoki` ライブラリを初期化します。
8. 署名、検証、および鍵ペア生成用の機構を備えたスロットを検索します。この作業では、`getMySlot()` という名前の関数を使って次の手順を実行します。

a. 関数 C_GetSlotList() を呼び出して利用可能なスロットのリストを取得します。

   PKCS #11 の規約でも推奨されているように、C_GetSlotList() は 2 回呼び出されます。1 回目の C_GetSlotList() 呼び出しでは、メモリーを割り当てる目的でスロット数を取得します。そして 2 回目の C_GetSlotList() 呼び出しでは、スロットを取得します。

b. 目的の機構を提供できるスロットを検索します。

   この関数はスロットごとに、GetMechanismInfo() を呼び出して署名用の機構と鍵ペア生成用の機構を検索します。それらの機構がそのスロットでサポートされていない場合、GetMechanismInfo() はエラーを返します。GetMechanismInfo() が成功を返した場合、機構のフラグを検査し、その機構が必要な処理を実行できることを確認します。

9. C_OpenSession() を呼び出してセッションをオープンします。
10. C_GenerateKeyPair() を使って鍵ペアを生成します。
11. C_GetAttributeValue() を使って公開鍵を表示します。デモ専用です。
12. 署名が C_SignInit() で開始され、C_Sign() で完了します。
13. 検証が C_VerifyInit() で開始され、C_Verify() で完了します。
14. セッションをクローズします。

    このプログラムは、C_CloseSession() を使ってセッションをクローズし、C_Finalize() を使ってライブラリをクローズします。

署名と検証のソースコード例を、次に示します。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

例 9–3　PKCS #11 関数によるテキストの署名と検証

```
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <security/cryptoki.h>
#include <security/pkcs11.h>

#define BUFFERSIZ    8192

/* Define key template */
static CK_BBOOL truevalue = TRUE;
static CK_BBOOL falsevalue = FALSE;
static CK_ULONG modulusbits = 1024;
static CK_BYTE public_exponent[] = {3};

boolean_t GetMySlot(CK_MECHANISM_TYPE sv_mech, CK_MECHANISM_TYPE kpgen_mech,
```

例 9–3 PKCS#11 関数によるテキストの署名と検証 (続き)

```
    CK_SLOT_ID_PTR pslot);

/* Example signs and verifies a simple string, using a public/private
 * key pair. */
void
main(int argc, char **argv)
{
    CK_RV   rv;
    CK_MECHANISM genmech, smech;
    CK_SESSION_HANDLE hSession;
    CK_SESSION_INFO sessInfo;
    CK_SLOT_ID slotID;
    int error, i = 0;

    CK_OBJECT_HANDLE privatekey, publickey;

    /* Set public key. */
    CK_ATTRIBUTE publickey_template[] = {
        {CKA_VERIFY, &truevalue, sizeof (truevalue)},
        {CKA_MODULUS_BITS, &modulusbits, sizeof (modulusbits)},
        {CKA_PUBLIC_EXPONENT, &public_exponent,
            sizeof (public_exponent)}
    };

    /* Set private key. */
    CK_ATTRIBUTE privatekey_template[] = {
        {CKA_SIGN, &truevalue, sizeof (truevalue)},
        {CKA_TOKEN, &falsevalue, sizeof (falsevalue)},
        {CKA_SENSITIVE, &truevalue, sizeof (truevalue)},
        {CKA_EXTRACTABLE, &truevalue, sizeof (truevalue)}
    };

    /* Create sample message. */
    CK_ATTRIBUTE getattributes[] = {
        {CKA_MODULUS_BITS, NULL_PTR, 0},
        {CKA_MODULUS, NULL_PTR, 0},
        {CKA_PUBLIC_EXPONENT, NULL_PTR, 0}
    };

    CK_ULONG messagelen, slen, template_size;

    boolean_t found_slot = B_FALSE;
    uchar_t *message = (uchar_t *)"Simple message for signing & verifying.";
    uchar_t *modulus, *pub_exponent;
    char sign[BUFFERSIZ];
    slen = BUFFERSIZ;

    messagelen = strlen((char *)message);

    /* Set up mechanism for generating key pair */
    genmech.mechanism = CKM_RSA_PKCS_KEY_PAIR_GEN;
    genmech.pParameter = NULL_PTR;
    genmech.ulParameterLen = 0;

    /* Set up the signing mechanism */
    smech.mechanism = CKM_RSA_PKCS;
```

例 9–3　PKCS #11 関数によるテキストの署名と検証　　(続き)

```
    smech.pParameter = NULL_PTR;
    smech.ulParameterLen = 0;

    /* Initialize the CRYPTOKI library */
    rv = C_Initialize(NULL_PTR);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_Initialize: Error = 0x%.8X\n", rv);
        exit(1);
    }

    found_slot = GetMySlot(smech.mechanism, genmech.mechanism, &slotID);

    if (!found_slot) {
        fprintf(stderr, "No usable slot was found.\n");
        goto exit_program;
    }

    fprintf(stdout, "selected slot: %d\n", slotID);

    /* Open a session on the slot found */
    rv = C_OpenSession(slotID, CKF_SERIAL_SESSION, NULL_PTR, NULL_PTR,
        &hSession);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_OpenSession: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_program;
    }

    fprintf(stdout, "Generating keypair....\n");

    /* Generate Key pair for signing/verifying */
    rv = C_GenerateKeyPair(hSession, &genmech, publickey_template,
        (sizeof (publickey_template) / sizeof (CK_ATTRIBUTE)),
        privatekey_template,
        (sizeof (privatekey_template) / sizeof (CK_ATTRIBUTE)),
        &publickey, &privatekey);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_GenerateKeyPair: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_session;
    }

    /* Display the publickey. */
    template_size = sizeof (getattributes) / sizeof (CK_ATTRIBUTE);

    rv = C_GetAttributeValue(hSession, publickey, getattributes,
        template_size);

    if (rv != CKR_OK) {
        /* not fatal, we can still sign/verify if this failed */
        fprintf(stderr, "C_GetAttributeValue: rv = 0x%.8X\n", rv);
        error = 1;
    } else {
```

例 9–3 PKCS#11関数によるテキストの署名と検証 (続き)

```
        /* Allocate memory to hold the data we want */
        for (i = 0; i < template_size; i++) {
            getattributes[i].pValue =
                malloc (getattributes[i].ulValueLen *
                sizeof(CK_VOID_PTR));
            if (getattributes[i].pValue == NULL) {
                int j;
                for (j = 0; j < i; j++)
                    free(getattributes[j].pValue);
                goto sign_cont;
            }
        }

        /* Call again to get actual attributes */
        rv = C_GetAttributeValue(hSession, publickey, getattributes,
            template_size);

        if (rv != CKR_OK) {
            /* not fatal, we can still sign/verify if failed */
            fprintf(stderr,
                "C_GetAttributeValue: rv = 0x%.8X\n", rv);
            error = 1;
        } else {
            /* Display public key values */
            fprintf(stdout, "Public Key data:\n\tModulus bits: "
                "%d\n",
                *((CK_ULONG_PTR)(getattributes[0].pValue)));

            fprintf(stdout, "\tModulus: ");
            modulus = (uchar_t *)getattributes[1].pValue;
            for (i = 0; i < getattributes[1].ulValueLen; i++) {
                fprintf(stdout, "%.2x", modulus[i]);
            }

            fprintf(stdout, "\n\tPublic Exponent: ");
            pub_exponent = (uchar_t *)getattributes[2].pValue;
            for (i = 0; i< getattributes[2].ulValueLen; i++) {
                fprintf(stdout, "%.2x", pub_exponent[i]);
            }
            fprintf(stdout, "\n");
        }
    }

sign_cont:
    rv = C_SignInit(hSession, &smech, privatekey);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_SignInit: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_session;
    }

    rv = C_Sign(hSession, (CK_BYTE_PTR)message, messagelen,
        (CK_BYTE_PTR)sign, &slen);

    if (rv != CKR_OK) {
```

例 9–3 PKCS #11 関数によるテキストの署名と検証 *(続き)*

```
        fprintf(stderr, "C_Sign: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_session;
    }

    fprintf(stdout, "Message was successfully signed with private key!\n");

    rv = C_VerifyInit(hSession, &smech, publickey);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_VerifyInit: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_session;
    }

    rv = C_Verify(hSession, (CK_BYTE_PTR)message, messagelen,
        (CK_BYTE_PTR)sign, slen);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_Verify: rv = 0x%.8X\n", rv);
        error = 1;
        goto exit_session;
    }

    fprintf(stdout, "Message was successfully verified with public key!\n");

exit_session:
    (void) C_CloseSession(hSession);

exit_program:
    (void) C_Finalize(NULL_PTR);

    for (i = 0; i < template_size; i++) {
        if (getattributes[i].pValue != NULL)
            free(getattributes[i].pValue);
    }

    exit(error);

}

/* Find a slot capable of:
 * . signing and verifying with sv_mech
 * . generating a key pair with kpgen_mech
 * Returns B_TRUE when successful. */
boolean_t GetMySlot(CK_MECHANISM_TYPE sv_mech, CK_MECHANISM_TYPE kpgen_mech,
    CK_SLOT_ID_PTR pSlotID)
{
    CK_SLOT_ID_PTR pSlotList = NULL_PTR;
    CK_SLOT_ID SlotID;
    CK_ULONG ulSlotCount = 0;
    CK_MECHANISM_INFO mech_info;
    int i;
    boolean_t returnval = B_FALSE;

    CK_RV rv;
```

例 9–3 PKCS#11 関数によるテキストの署名と検証 (続き)

```
/* Get slot list for memory alloction */
rv = C_GetSlotList(0, NULL_PTR, &ulSlotCount);

if ((rv == CKR_OK) && (ulSlotCount > 0)) {
    fprintf(stdout, "slotCount = %d\n", ulSlotCount);
    pSlotList = malloc(ulSlotCount * sizeof (CK_SLOT_ID));

    if (pSlotList == NULL) {
        fprintf(stderr, "System error: unable to allocate "
            "memory\n");
        return (returnval);
    }

    /* Get the slot list for processing */
    rv = C_GetSlotList(0, pSlotList, &ulSlotCount);
    if (rv != CKR_OK) {
        fprintf(stderr, "GetSlotList failed: unable to get "
            "slot count.\n");
        goto cleanup;
    }
} else {
    fprintf(stderr, "GetSlotList failed: unable to get slot "
        "list.\n");
    return (returnval);
}

/* Find a slot capable of specified mechanism */
for (i = 0; i < ulSlotCount; i++) {
    SlotID = pSlotList[i];

    /* Check if this slot is capable of signing and
     * verifying with sv_mech. */
    rv = C_GetMechanismInfo(SlotID, sv_mech, &mech_info);

    if (rv != CKR_OK) {
        continue;
    }

    if (!(mech_info.flags & CKF_SIGN &&
        mech_info.flags & CKF_VERIFY)) {
        continue;
    }

    /* Check if the slot is capable of key pair generation
     * with kpgen_mech. */
    rv = C_GetMechanismInfo(SlotID, kpgen_mech, &mech_info);

    if (rv != CKR_OK) {
        continue;
    }

    if (!(mech_info.flags & CKF_GENERATE_KEY_PAIR)) {
        continue;
    }
```

例 9–3　PKCS #11 関数によるテキストの署名と検証　　(続き)

```
        /* If we get this far, this slot supports our mechanisms. */
        returnval = B_TRUE;
        *pSlotID = SlotID;
        break;
    }

cleanup:
    if (pSlotList)
        free(pSlotList);
    return (returnval);
}
```

ランダムバイト生成の例

例 9–4 は、ランダムバイトを生成できる機構を備えたスロットを検索する方法を示しています。この例では、次の手順を実行します。

1. cryptoki ライブラリを初期化します。
2. GetRandSlot() を呼び出して、ランダムバイトを生成できる機構を備えたスロットを検索します。

   この作業におけるスロット検索手順は次のとおりです。

   a. 関数 C_GetSlotList() を呼び出して利用可能なスロットのリストを取得します。

      PKCS #11 の規約で推奨されているように、C_GetSlotList() は 2 回呼び出されます。1 回目の C_GetSlotList() 呼び出しでは、メモリーを割り当てる目的でスロット数を取得します。そして 2 回目の C_GetSlotList() 呼び出しでは、スロットを取得します。

   b. ランダムバイトを生成できるスロットを検索します。

      この関数はスロットごとに、GetTokenInfo() を使ってトークン情報を取得し、CKF_RNG フラグセットを含むエントリの有無を検査します。CKF_RNG フラグセットを含むスロットが見つかった場合、GetRandSlot() 関数が戻ります。

3. C_OpenSession() を使ってセッションをオープンします。
4. C_GenerateRandom() を使ってランダムバイトを生成します。
5. セッションを終了します。

   このプログラムは、C_CloseSession() を使ってセッションをクローズし、C_Finalize() を使ってライブラリをクローズします。

乱数生成のソースコード例を、次に示します。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

例 9–4　PKCS #11 関数による乱数生成

```
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <security/cryptoki.h>
#include <security/pkcs11.h>

#define RANDSIZE 64

boolean_t GetRandSlot(CK_SLOT_ID_PTR pslot);

/* Example generates random bytes. */
void
main(int argc, char **argv)
{
    CK_RV   rv;
    CK_MECHANISM mech;
    CK_SESSION_HANDLE hSession;
    CK_SESSION_INFO sessInfo;
    CK_SLOT_ID slotID;
    CK_BYTE randBytes[RANDSIZE];

    boolean_t found_slot = B_FALSE;
    int error;
    int i;

    /* Initialize the CRYPTOKI library */
    rv = C_Initialize(NULL_PTR);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_Initialize: Error = 0x%.8X\n", rv);
        exit(1);
    }

    found_slot = GetRandSlot(&slotID);

    if (!found_slot) {
        goto exit_program;
    }

    /* Open a session on the slot found */
    rv = C_OpenSession(slotID, CKF_SERIAL_SESSION, NULL_PTR, NULL_PTR,
        &hSession);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_OpenSession: rv = 0x%.8x\n", rv);
        error = 1;
        goto exit_program;
    }
```

例 9–4　PKCS #11 関数による乱数生成　　(続き)

```
    /* Generate random bytes */
    rv = C_GenerateRandom(hSession, randBytes, RANDSIZE);

    if (rv != CKR_OK) {
        fprintf(stderr, "C_GenerateRandom: rv = 0x%.8x\n", rv);
        error = 1;
        goto exit_session;
    }

    fprintf(stdout, "Random value: ");
    for (i = 0; i < RANDSIZE; i++) {
        fprintf(stdout, "%.2x", randBytes[i]);
    }

exit_session:
    (void) C_CloseSession(hSession);

exit_program:
    (void) C_Finalize(NULL_PTR);
    exit(error);

}

boolean_t
GetRandSlot(CK_SLOT_ID_PTR pslot)
{
    CK_SLOT_ID_PTR pSlotList;
    CK_SLOT_ID SlotID;
    CK_TOKEN_INFO tokenInfo;
    CK_ULONG ulSlotCount;
    CK_MECHANISM_TYPE_PTR pMechTypeList = NULL_PTR;
    CK_ULONG ulMechTypecount;
    boolean_t result = B_FALSE;
    int i = 0;

    CK_RV rv;

    /* Get slot list for memory allocation */
    rv = C_GetSlotList(0, NULL_PTR, &ulSlotCount);

    if ((rv == CKR_OK) && (ulSlotCount > 0)) {
        fprintf(stdout, "slotCount = %d\n", (int)ulSlotCount);
        pSlotList = malloc(ulSlotCount * sizeof (CK_SLOT_ID));

        if (pSlotList == NULL) {
            fprintf(stderr,
                "System error: unable to allocate memory\n");
            return (result);
        }

        /* Get the slot list for processing */
        rv = C_GetSlotList(0, pSlotList, &ulSlotCount);
        if (rv != CKR_OK) {
            fprintf(stderr, "GetSlotList failed: unable to get "
                "slot list.\n");
            free(pSlotList);
```

例 9–4 PKCS#11 関数による乱数生成 (続き)

```
            return (result);
        }
    } else {
        fprintf(stderr, "GetSlotList failed: unable to get slot"
            " count.\n");
        return (result);
    }

    /* Find a slot capable of doing random number generation */
    for (i = 0; i < ulSlotCount; i++) {
        SlotID = pSlotList[i];

        rv = C_GetTokenInfo(SlotID, &tokenInfo);

        if (rv != CKR_OK) {
            /* Check the next slot */
            continue;
        }

        if (tokenInfo.flags & CKF_RNG) {
            /* Found a random number generator */
            *pslot = SlotID;
            fprintf(stdout, "Slot # %d supports random number "
                "generation!\n", SlotID);
            result = B_TRUE;
            break;
        }
    }

    if (pSlotList)
        free(pSlotList);

    return (result);

}
```

ユーザーレベルのプロバイダの例

ユーザーレベルのプロバイダの例については、OpenSolaris の `pkcs11_softtoken` の例を参照してください。これは実際に動作するプロバイダで、独自のプロバイダを作成するときの基盤として使用できます。

このプロバイダのコードを利用するには、OpenSolaris の Web サイト `http://src.opensolaris.org/source/` にアクセスします。「File Path」フィールドに「`pkcs11_softtoken/common`」と入力し、`onnv` プロジェクトが選択されていることを確認し、「Search」ボタンをクリックします。

◆ ◆ ◆ 第 10 章

10 スマートカードフレームワークの使用

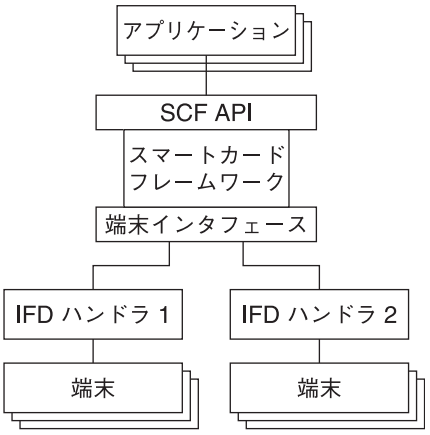
スマートカードとは、マイクロプロセッサとメモリーを備えたポータブルコンピュータのことです。通常、スマートカードの形状と大きさはクレジットカードと同じです。スマートカードを使用すると、認証や暗号化を通じて保護される機密情報を非常に安全に保存できます。

この章で扱う内容は、次のとおりです。

Oracle Solaris スマートカードフレームワークの概要

Solaris オペレーティングシステムでは、スマートカード端末を持つコンシューマアプリケーションへの接続にスマートカードフレームワークが使用されます。コンシューマアプリケーションは、スマートカードフレームワーク (SCF) API の呼び出しを行います。スマートカード端末は、基本的にはデバイスドライバであるインタフェースデバイス (IFD) ハンドラを介してコンシューマアプリケーションとのやりとりを行います。IFD ハンドラは、端末インタフェースを通じてフレームワークに接続します。次の図を参照してください。

図 10–1 スマートカードフレームワーク



Solaris オペレーティングシステムでは、スマートカードの構成情報を非公開ファイルに格納します。この方法は、通常 `/etc/reader.conf` が使用される Linux の実装とは対照的です。構成ファイルの項目を変更するには、コマンド `smartcard(1M)` を使用します。

現時点では、スマートカードフレームワークは Solaris 暗号化フレームワークに依存していません。

スマートカードのコンシューマアプリケーションの開発

SCF API には、スマートカードにアクセスするための 1 組のインタフェースが用意されています。これらのインタフェースを使用すると、低レベルのアプリケーションプロトコルデータユニット (APDU) 形式でスマートカードとの通信を行うことができます。これらのインタフェースは、C と Java の両方で提供されます。これらのインタフェースは、Solaris オペレーティングシステムでサポートされているすべてのリーダー、および APDU で通信を行うすべてのスマートカードで動作します。SCF API は、次のコンポーネントに基づいています。

- セッションオブジェクト – 衝突を避けるための個々のスレッドの汎用コンテキスト。
- 端末オブジェクト – 実際のスマートカード端末を抽象化したもの。このオブジェクトは、カードの存在、挿入、または取り外しを検出できます。
- カードオブジェクト – 端末に挿入されるスマートカードを表します。このオブジェクトは、APDU 形式の情報を実際のスマートカードに送信します。また、アプリケーションがカードへの排他的アクセスを行えるように相互排他ロックにも対応しています。
- リスナーオブジェクト – イベントの通知を受信するオブジェクト。

SCF API には、次のような機能が備わっています。

- スマートカードがリーダーに物理的に存在するかどうか検査する。
- スマートカードの動作 (挿入や取り外し) の通知を受信する。
- データをスマートカードと交換する。
- セッション、端末、およびスマートカードに関する情報を取り出す。
- 排他的アクセスに備えてスマートカードをロックおよびロック解除する。

次の節では、それぞれの SCF インタフェースについて説明します。

SCF セッションインタフェース

SCF セッションには、次の関数が使用されます。

`SCF_Session_getSession(3SMARTCARD)`
システムのスマートカードフレームワークを使ってセッションを確立します。セッションが開いたら、`SCF_Session_getTerminal(3SMARTCARD)` を使ってスマートカード端末にアクセスできます。

`SCF_Session_close(3SMARTCARD)`
セッションが開いたときに割り当てられたリソースを解放します。また、そのセッションに関連付けられている端末またはカードをすべて閉じます。

`SCF_Session_getInfo(3SMARTCARD)`
セッションに関する情報を取得します。

`SCF_Session_freeInfo(3SMARTCARD)`
`SCF_Session_getInfo(3SMARTCARD)` によって戻された記憶領域を解放します。

`SCF_Session_getTerminal(3SMARTCARD)`
セッション内で特定のスマートカード端末によるコンテキストを確立します。カードの動作 (挿入または取り外し) の検出には、端末オブジェクトが使用されます。端末オブジェクトは、特定のカードにアクセスするためのカードオブジェクトの作成にも使用されます。

SCF 端末インタフェース

SCF 端末のアクセスには、次の関数が使用されます。

`SCF_Terminal_close(3SMARTCARD)`
端末が開いたときに割り当てられたリソースを解放します。また、端末に関連付けられているカードもすべて閉じます。

`SCF_Terminal_getInfo(3SMARTCARD)`
端末に関する情報を取得します。

`SCF_Terminal_freeInfo(3SMARTCARD)`
`SCF_Terminal_getInfo(3SMARTCARD)` によって戻された記憶領域を解放します。

SCF_Terminal_waitForCardPresent(3SMARTCARD)
特定の端末にカードが装着されるまでブロックおよび待機します。

SCF_Terminal_waitForCardAbsent(3SMARTCARD)
特定の端末に装着されているカードが取り外されるまでブロックおよび待機します。

SCF_Terminal_addEventListener(3SMARTCARD)
端末でイベントが発生したときにコールバック通知をプログラムで受信できるようにします。シグナルハンドラと同じような働きです。イベントが発生すると、サービススレッドによって用意されたコールバック関数が実行されます。

SCF_Terminal_updateEventListener(3SMARTCARD)
この端末に関連付けられている指定のイベントリスナーを更新します。

SCF_Terminal_removeEventListener(3SMARTCARD)
この端末に関連付けられている指定のイベントリスナーをリスナーリストから削除します。

SCF_Terminal_getCard(3SMARTCARD)
端末内で特定のスマートカードによるコンテキストを確立します。SCF_Card_exchangeAPDU(3SMARTCARD) で APDU をカードに送信する際にはカードオブジェクトが使用されます。

SCF カードとその他のインタフェース

スマートカードへのアクセスと状態の取得には、次の関数が使用されます。

SCF_Card_close(3SMARTCARD)
カードが開いたときに割り当てられたメモリーやスレッドなどのリソースを解放します。また、そのカードによって保持されていたロックも解放します。

SCF_Card_getInfo(3SMARTCARD)
カードに関する情報を取得します。

SCF_Card_freeInfo(3SMARTCARD)
SCF_Card_getInfo(3SMARTCARD) によって戻された記憶領域を解放します。

SCF_Card_lock(3SMARTCARD)
特定のカードに対するロックを取得します。この関数を使用すると、アプリケーションはほかのスマートカードアプリケーションからの干渉を受けないで、多重 APDU トランザクションを実行できます。

SCF_Card_unlock(3SMARTCARD)
特定のカードからロックを削除します。

SCF_Card_exchangeAPDU(3SMARTCARD)
コマンド APDU をカードに送信し、カードの応答を読み取ります。

SCF_Card_waitForCardRemoved(3SMARTCARD)
: 特定のカードが削除されたかどうかを確認します。削除された後で別のカードまたは同じカードが再度挿入された場合は、古いカードが削除されたことを報告します。

SCF_Card_reset(3SMARTCARD)
: 特定のカードをリセットします。

SCF_strerror(3SMARTCARD)
: 状態コードを記述する文字列を取得します。

スマートカード端末用の IFD ハンドラの開発

Solaris OS 用に開発されるスマートカード端末では、Linux のスマートカード端末で使用されるのと同じ API セットが使用されます。以前に IFD ハンドラを開発したことがない場合は、IFD ソースコードを提供する Linux 環境用のいずれかの Web サイト (http://www.musclecard.com/drivers.html など) を参照したほうがよいでしょう。Solaris オペレーティングシステムでスマートカード端末用の IFD ハンドラを開発するには、 `/usr/include/smartcard/ifdhandler.h` をインクルードし、次のインタフェースを実装する必要があります。

- IFDHCreateChannelByName(3SMARTCARD) – 特定のスマートカード端末との通信チャネルを開きます。このインタフェースは、最新バージョンの MUSCLE IFD 仕様の新機能です。このため、`IFDHCreateChannelByName()` はほかの IFD ハンドラでは使用できません。Solaris ソフトウェアでは、IFDHCreateChannel(3SMARTCARD) 関数の代わりに `IFDHCreateChannelByName ()` が使用されます。
- IFDHICCPresence(3SMARTCARD) – リーダー、または論理ユニット番号 (LUN) によって指定されたスロットに ICC (スマートカード) が存在するかどうかを確認します。
- IFDHPowerICC(3SMARTCARD) – ICC の電源とリセットの信号を制御します。
- IFDHCloseChannel(3SMARTCARD) – LUN によって指定された IFD の通信チャネルを閉じます。
- IFDHGetCapabilities(3SMARTCARD) – 指定されたスマートカード、IFD ハンドラ、またはスマートカード端末の性能を戻します。
- IFDHSetProtocolParameters(3SMARTCARD) – 特定のスロットまたはカードに対して PTS (Protocol Type Selection) を設定します。PTS の値は、ISO 7816 標準を確認してください。この関数は、フレームワークによって呼び出されないかもしれませんが、実装すべきです。`IFDHSetProtocolParameters()` を使用して、各種カードが確実にフレームワークと通信できることを確認してください。
- IFDHTransmitToICC(3SMARTCARD) – スマートカードと通信するためにフレームワークによって呼び出されます。

注 – IFDHCreateChannel()、IFDHSetCapabilities()、および IFDHControl() は現在使用されていませんが、これらのインタフェースは今後のリリースで必要になる場合があります。

IFDHICCPresence() および IFDHPowerICC() の各関数はテストの際に役に立ちます。たとえば、IFDHICCPresence() 関数を使用して、スロット内にカードが存在するかどうかをテストできます。スマートカードの電源が正常に機能しているかどうかを確認する場合は、IFDHPowerICC() 関数を使用するのも 1 つの方法です。この関数は、挿入されているスマートカードの ATR (Answer to Reset) 値を取得します。

スマートカード端末のインストール

Solaris スマートカードフレームワークは、USB 端末などのホットプラグ可能な端末には対応していません。スマートカード端末に接続してインストールするには、次の手順を実行してください。

1. スマートカード端末をシステムに物理的に接続します。
2. IFD ハンドラ用の共有ライブラリをシステムにコピーします。
3. smartcard(1M) を使用して、端末用の IFD ハンドラをフレームワーク内に登録します。

◆ ◆ ◆ 付 録 A

C ベースの GSS-API プログラム例

この付録では、GSS-API を使用して安全なネットワーク接続を行う 2 つのアプリケーションのソースコード例を示します。最初のアプリケーションは、一般的なクライアントアプリケーションです。2 つ目のアプリケーションは、GSS-API でサーバーがどのように動作するかを示すものです。2 つのプログラムは、実行中にベンチマークを表示します。このため、ユーザーは GSS-API が動作しているのを見ることができます。さらに、クライアントアプリケーションとサーバーアプリケーションが使用する補助的な関数もいくつか示します。

各プログラムの詳細は、第 5 章「GSS-API クライアント例」および第 6 章「GSS-API サーバー例」を参照してください。

クライアント側アプリケーション

次の例では、クライアント側プログラム `gss_client` のソースコードを示します。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。`http://www.sun.com/download/products.xml?id=41912db5` を参照してください。

例 A–1 `gss-client.c` プログラム例の完全なリスト

```
/*
 * Copyright 1994 by OpenVision Technologies, Inc.
 *
 * Permission to use, copy, modify, distribute, and sell this software
 * and its documentation for any purpose is hereby granted without fee,
 * provided that the above copyright notice appears in all copies and
```

例 A–1 gss-client.c プログラム例の完全なリスト (続き)

```
 * that both that copyright notice and this permission notice appear in
 * supporting documentation, and that the name of OpenVision not be used
 * in advertising or publicity pertaining to distribution of the software
 * without specific, written prior permission. OpenVision makes no
 * representations about the suitability of this software for any
 * purpose.  It is provided "as is" without express or implied warranty.
 *
 * OPENVISION DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
 * INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO
 * EVENT SHALL OPENVISION BE LIABLE FOR ANY SPECIAL, INDIRECT OR
 * CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
 * USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
 * OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
 * PERFORMANCE OF THIS SOFTWARE.
 */

#if !defined(lint) && !defined(__CODECENTER__)
static char *rcsid = \
"$Header: /cvs/krbdev/krb5/src/appl/gss-sample/gss-client.c,\
v 1.16 1998/10/30 02:52:03 marc Exp $";
#endif

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <errno.h>
#include <sys/stat.h>
#include <fcntl.h>

#include <gssapi/gssapi.h>
#include <gssapi/gssapi_ext.h>
#include <gss-misc.h>

void usage()
{
     fprintf(stderr, "Usage: gss-client [-port port] [-d] host service \
msg\n");
     exit(1);
}

/*
 * Function: connect_to_server
 *
 * Purpose: Opens a TCP connection to the name host and port.
 *
 * Arguments:
 *
 *      host            (r) the target host name
 *      port            (r) the target port, in host byte order
 *
```

例 A–1 gss-client.c プログラム例の完全なリスト (続き)

```
 * Returns: the established socket file descriptor, or -1 on failure
 *
 * Effects:
 *
 * The host name is resolved with gethostbyname(), and the socket is
 * opened and connected.  If an error occurs, an error message is
 * displayed and -1 is returned.
 */
int connect_to_server(host, port)
     char *host;
     u_short port;
{
     struct sockaddr_in saddr;
     struct hostent *hp;
     int s;

     if ((hp = gethostbyname(host)) == NULL) {
          fprintf(stderr, "Unknown host: %s\n", host);
          return -1;
     }

     saddr.sin_family = hp->h_addrtype;
     memcpy((char *)&saddr.sin_addr, hp->h_addr, sizeof(saddr.sin_addr));
     saddr.sin_port = htons(port);

     if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
          perror("creating socket");
          return -1;
     }
     if (connect(s, (struct sockaddr *)&saddr, sizeof(saddr)) < 0) {
          perror("connecting to server");
          (void) close(s);
          return -1;
     }
     return s;
}

/*
 * Function: client_establish_context
 *
 * Purpose: establishes a GSS-API context with a specified service and
 * returns the context handle
 *
 * Arguments:
 *
 *      s               (r) an established TCP connection to the service
 *      service_name    (r) the ASCII service name of the service
 *      context         (w) the established GSS-API context
 *      ret_flags       (w) the returned flags from init_sec_context
 *
 * Returns: 0 on success, -1 on failure
 *
 * Effects:
 *
 * service_name is imported as a GSS-API name and a GSS-API context is
 * established with the corresponding service; the service should be
```

例 A–1 gss-client.c プログラム例の完全なリスト (続き)

```
 * listening on the TCP connection s.  The default GSS-API mechanism
 * is used, and mutual authentication and replay detection are
 * requested.
 *
 * If successful, the context handle is returned in context.  If
 * unsuccessful, the GSS-API error messages are displayed on stderr
 * and -1 is returned.
 */
int client_establish_context(s, service_name, deleg_flag, oid,
                             gss_context, ret_flags)
     int s;
     char *service_name;
     gss_OID oid;
     OM_uint32 deleg_flag;
     gss_ctx_id_t *gss_context;
     OM_uint32 *ret_flags;
{
     gss_buffer_desc send_tok, recv_tok, *token_ptr;
     gss_name_t target_name;
     OM_uint32 maj_stat, min_stat, init_sec_min_stat;

     /*
      * Import the name into target_name.  Use send_tok to save
      * local variable space.
      */
     send_tok.value = service_name;
     send_tok.length = strlen(service_name) + 1;
     maj_stat = gss_import_name(&min_stat, &send_tok,
         (gss_OID) GSS_C_NT_HOSTBASED_SERVICE, &target_name);
     if (maj_stat != GSS_S_COMPLETE) {
          display_status("parsing name", maj_stat, min_stat);
          return -1;
     }

     /*
      * Perform the context-establishement loop.
      *
      * On each pass through the loop, token_ptr points to the token
      * to send to the server (or GSS_C_NO_BUFFER on the first pass).
      * Every generated token is stored in send_tok which is then
      * transmitted to the server; every received token is stored in
      * recv_tok, which token_ptr is then set to, to be processed by
      * the next call to gss_init_sec_context.
      *
      * GSS-API guarantees that send_tok's length will be non-zero
      * if and only if the server is expecting another token from us,
      * and that gss_init_sec_context returns GSS_S_CONTINUE_NEEDED if
      * and only if the server has another token to send us.
      */

     token_ptr = GSS_C_NO_BUFFER;
     *gss_context = GSS_C_NO_CONTEXT;

     do {
          maj_stat =
               gss_init_sec_context(&init_sec_min_stat,
```

例 A–1 gss-client.cプログラム例の完全なリスト (続き)

```
                                    GSS_C_NO_CREDENTIAL,
                                    gss_context,
                                    target_name,
                                    oid,
                                    GSS_C_MUTUAL_FLAG | GSS_C_REPLAY_FLAG |
                                                        deleg_flag,
                                    0,
                                    NULL,       /* no channel bindings */
                                    token_ptr,
                                    NULL,       /* ignore mech type */
                                    &send_tok,
                                    ret_flags,
                                    NULL);      /* ignore time_rec */

          if (token_ptr != GSS_C_NO_BUFFER)
               (void) gss_release_buffer(&min_stat, &recv_tok);

          if (send_tok.length != 0) {
               printf("Sending init_sec_context token (size=%d)...",
                     send_tok.length);
               if (send_token(s, &send_tok) < 0) {
                    (void) gss_release_buffer(&min_stat, &send_tok);
                    (void) gss_release_name(&min_stat, &target_name);
                    return -1;
               }
          }
          (void) gss_release_buffer(&min_stat, &send_tok);

          if (maj_stat!=GSS_S_COMPLETE && maj_stat!=GSS_S_CONTINUE_NEEDED) {
               display_status("initializing context", maj_stat,
                              init_sec_min_stat);
               (void) gss_release_name(&min_stat, &target_name);
               if (*gss_context == GSS_C_NO_CONTEXT)
                       gss_delete_sec_context(&min_stat, gss_context,
                                              GSS_C_NO_BUFFER);
               return -1;
          }

          if (maj_stat == GSS_S_CONTINUE_NEEDED) {
               printf("continue needed...");
               if (recv_token(s, &recv_tok) < 0) {
                    (void) gss_release_name(&min_stat, &target_name);
                    return -1;
               }
               token_ptr = &recv_tok;
          }
          printf("\n");
     } while (maj_stat == GSS_S_CONTINUE_NEEDED);

     (void) gss_release_name(&min_stat, &target_name);
     return 0;
}

void read_file(file_name, in_buf)
    char                *file_name;
    gss_buffer_t        in_buf;
```

例 A–1 gss-client.c プログラム例の完全なリスト (続き)

```
{
    int fd, bytes_in, count;
    struct stat stat_buf;

    if ((fd = open(file_name, O_RDONLY, 0)) < 0) {
        perror("open");
        fprintf(stderr, "Couldn't open file %s\n", file_name);
        exit(1);
    }
    if (fstat(fd, &stat_buf) < 0) {
        perror("fstat");
        exit(1);
    }
    in_buf->length = stat_buf.st_size;

    if (in_buf->length == 0) {
        in_buf->value = NULL;
        return;
    }

    if ((in_buf->value = malloc(in_buf->length)) == 0) {
        fprintf(stderr, \
            "Couldn't allocate %d byte buffer for reading file\n",
            in_buf->length);
        exit(1);
    }

    /* this code used to check for incomplete reads, but you can't get
       an incomplete read on any file for which fstat() is meaningful */

    count = read(fd, in_buf->value, in_buf->length);
    if (count < 0) {
        perror("read");
        exit(1);
    }
    if (count < in_buf->length)
        fprintf(stderr, "Warning, only read in %d bytes, expected %d\n",
                count, in_buf->length);
}

/*
 * Function: call_server
 *
 * Purpose: Call the "sign" service.
 *
 * Arguments:
 *
 *      host            (r) the host providing the service
 *      port            (r) the port to connect to on host
 *      service_name    (r) the GSS-API service name to authenticate to
 *      msg             (r) the message to have "signed"
 *
 * Returns: 0 on success, -1 on failure
 *
 * Effects:
 *
```

例 A–1 gss-client.cプログラム例の完全なリスト (続き)

```
 * call_server opens a TCP connection to <host:port> and establishes a
 * GSS-API context with service_name over the connection.  It then
 * seals msg in a GSS-API token with gss_seal, sends it to the server,
 * reads back a GSS-API signature block for msg from the server, and
 * verifies it with gss_verify.  -1 is returned if any step fails,
 * otherwise 0 is returned.  */
int call_server(host, port, oid, service_name, deleg_flag, msg, use_file)
     char *host;
     u_short port;
     gss_OID oid;
     char *service_name;
     OM_uint32 deleg_flag;
     char *msg;
     int use_file;
{
     gss_ctx_id_t context;
     gss_buffer_desc in_buf, out_buf;
     int s, state;
     OM_uint32 ret_flags;
     OM_uint32 maj_stat, min_stat;
     gss_name_t         src_name, targ_name;
     gss_buffer_desc    sname, tname;
     OM_uint32          lifetime;
     gss_OID            mechanism, name_type;
     int                is_local;
     OM_uint32          context_flags;
     int                is_open;
     gss_qop_t          qop_state;
     gss_OID_set        mech_names;
     gss_buffer_desc    oid_name;
     size_t     i;

     /* Open connection */
     if ((s = connect_to_server(host, port)) < 0)
          return -1;

     /* Establish context */
     if (client_establish_context(s, service_name, deleg_flag, oid,
         &context, &ret_flags) < 0) {
         (void) close(s);
         return -1;
     }

     /* display the flags */
     display_ctx_flags(ret_flags);

     /* Get context information */
     maj_stat = gss_inquire_context(&min_stat, context,
                                    &src_name, &targ_name, &lifetime,
                                    &mechanism, &context_flags,
                                    &is_local,
                                    &is_open);
     if (maj_stat != GSS_S_COMPLETE) {
         display_status("inquiring context", maj_stat, min_stat);
         return -1;
     }
```

例 A–1 gss-client.c プログラム例の完全なリスト (続き)

```
maj_stat = gss_display_name(&min_stat, src_name, &sname,
                            &name_type);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("displaying source name", maj_stat, min_stat);
    return -1;
}
maj_stat = gss_display_name(&min_stat, targ_name, &tname,
                            (gss_OID *) NULL);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("displaying target name", maj_stat, min_stat);
    return -1;
}
fprintf(stderr, "\"%.*s\" to \"%.*s\", lifetime %d, flags %x, %s,
        %s\n", (int) sname.length, (char *) sname.value,
        (int) tname.length, (char *) tname.value, lifetime,
        context_flags,
        (is_local) ? "locally initiated" : "remotely initiated",
        (is_open) ? "open" : "closed");

(void) gss_release_name(&min_stat, &src_name);
(void) gss_release_name(&min_stat, &targ_name);
(void) gss_release_buffer(&min_stat, &sname);
(void) gss_release_buffer(&min_stat, &tname);

maj_stat = gss_oid_to_str(&min_stat,
                          name_type,
                          &oid_name);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("converting oid->string", maj_stat, min_stat);
    return -1;
}
fprintf(stderr, "Name type of source name is %.*s.\n",
        (int) oid_name.length, (char *) oid_name.value);
(void) gss_release_buffer(&min_stat, &oid_name);

/* Now get the names supported by the mechanism */
maj_stat = gss_inquire_names_for_mech(&min_stat,
                                      mechanism,
                                      &mech_names);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("inquiring mech names", maj_stat, min_stat);
    return -1;
}

maj_stat = gss_oid_to_str(&min_stat,
                          mechanism,
                          &oid_name);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("converting oid->string", maj_stat, min_stat);
    return -1;
}
fprintf(stderr, "Mechanism %.*s supports %d names\n",
        (int) oid_name.length, (char *) oid_name.value,
        mech_names->count);
(void) gss_release_buffer(&min_stat, &oid_name);
```

例 A–1 gss-client.cプログラム例の完全なリスト (続き)

```
for (i=0; i<mech_names->count; i++) {
    maj_stat = gss_oid_to_str(&min_stat,
                              &mech_names->elements[i],
                              &oid_name);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("converting oid->string", maj_stat, min_stat);
        return -1;
    }
    fprintf(stderr, "  %d: %.*s\n", i,
            (int) oid_name.length, (char *) oid_name.value);

    (void) gss_release_buffer(&min_stat, &oid_name);
}
(void) gss_release_oid_set(&min_stat, &mech_names);

if (use_file) {
    read_file(msg, &in_buf);
} else {
    /* Seal the message */
    in_buf.value = msg;
    in_buf.length = strlen(msg);
}

maj_stat = gss_wrap(&min_stat, context, 1, GSS_C_QOP_DEFAULT,
                    &in_buf, &state, &out_buf);
if (maj_stat != GSS_S_COMPLETE) {
     display_status("sealing message", maj_stat, min_stat);
     (void) close(s);
     (void) gss_delete_sec_context(&min_stat, &context,
         GSS_C_NO_BUFFER);
     return -1;
} else if (! state) {
     fprintf(stderr, "Warning!  Message not encrypted.\n");
}

/* Send to server */
if (send_token(s, &out_buf) < 0) {
     (void) close(s);
     (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
     return -1;
}
(void) gss_release_buffer(&min_stat, &out_buf);

/* Read signature block into out_buf */
if (recv_token(s, &out_buf) < 0) {
     (void) close(s);
     (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
     return -1;
}

/* Verify signature block */
maj_stat = gss_verify_mic(&min_stat, context, &in_buf,
                          &out_buf, &qop_state);
if (maj_stat != GSS_S_COMPLETE) {
     display_status("verifying signature", maj_stat, min_stat);
```

例 A–1 gss-client.c プログラム例の完全なリスト (続き)

```
          (void) close(s);
          (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
          return -1;
     }
     (void) gss_release_buffer(&min_stat, &out_buf);

     if (use_file)
         free(in_buf.value);

     printf("Signature verified.\n");

     /* Delete context */
     maj_stat = gss_delete_sec_context(&min_stat, &context, &out_buf);
     if (maj_stat != GSS_S_COMPLETE) {
          display_status("deleting context", maj_stat, min_stat);
          (void) close(s);
          (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
          return -1;
     }

     (void) gss_release_buffer(&min_stat, &out_buf);
     (void) close(s);
     return 0;
}

static void parse_oid(char *mechanism, gss_OID *oid)
{
    char        *mechstr = 0, *cp;
    gss_buffer_desc tok;
    OM_uint32 maj_stat, min_stat;

    if (isdigit(mechanism[0])) {
        mechstr = malloc(strlen(mechanism)+5);
        if (!mechstr) {
            printf("Couldn't allocate mechanism scratch!\n");
            return;
        }
        sprintf(mechstr, "{ %s }", mechanism);
        for (cp = mechstr; *cp; cp++)
            if (*cp == '.')
                *cp = ' ';
        tok.value = mechstr;
    } else
        tok.value = mechanism;
    tok.length = strlen(tok.value);
    maj_stat = gss_str_to_oid(&min_stat, &tok, oid);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("str_to_oid", maj_stat, min_stat);
        return;
    }
    if (mechstr)
        free(mechstr);
}

int main(argc, argv)
     int argc;
```

例 A–1 gss-client.cプログラム例の完全なリスト (続き)

```
     char **argv;
{
     char *service_name, *server_host, *msg;
     char *mechanism = 0;
     u_short port = 4444;
     int use_file = 0;
     OM_uint32 deleg_flag = 0, min_stat;
     gss_OID oid = GSS_C_NULL_OID;

     display_file = stdout;

     /* Parse arguments. */
     argc--; argv++;
     while (argc) {
          if (strcmp(*argv, "-port") == 0) {
               argc--; argv++;
               if (!argc) usage();
               port = atoi(*argv);
          } else if (strcmp(*argv, "-mech") == 0) {
               argc--; argv++;
               if (!argc) usage();
               mechanism = *argv;
          } else if (strcmp(*argv, "-d") == 0) {
               deleg_flag = GSS_C_DELEG_FLAG;
          } else if (strcmp(*argv, "-f") == 0) {
               use_file = 1;
          } else
               break;
          argc--; argv++;
     }
     if (argc != 3)
          usage();

     server_host = *argv++;
     service_name = *argv++;
     msg = *argv++;

     if (mechanism)
         parse_oid(mechanism, &oid);

     if (call_server(server_host, port, oid, service_name,
                     deleg_flag, msg, use_file) < 0)
          exit(1);

     if (oid != GSS_C_NULL_OID)
         (void) gss_release_oid(&min_stat, &oid);

     return 0;
}
```

サーバー側アプリケーション

次の例では、サーバー側プログラム gss_server のソースコードを示します。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

例 A–2 gss-server.c プログラム例の完全なコードリスト

```
/*
 * Copyright 1994 by OpenVision Technologies, Inc.
 *
 * Permission to use, copy, modify, distribute, and sell this software
 * and its documentation for any purpose is hereby granted without fee,
 * provided that the above copyright notice appears in all copies and
 * that both that copyright notice and this permission notice appear in
 * supporting documentation, and that the name of OpenVision not be used
 * in advertising or publicity pertaining to distribution of the software
 * without specific, written prior permission. OpenVision makes no
 * representations about the suitability of this software for any
 * purpose.  It is provided "as is" without express or implied warranty.
 *
 * OPENVISION DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
 * INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO
 * EVENT SHALL OPENVISION BE LIABLE FOR ANY SPECIAL, INDIRECT OR
 * CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
 * USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
 * OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
 * PERFORMANCE OF THIS SOFTWARE.
 */

#if !defined(lint) && !defined(__CODECENTER__)
static char *rcsid = \
"$Header: /cvs/krbdev/krb5/src/appl/gss-sample/gss-server.c, \
    v 1.21 1998/12/22 \
04:10:08 tytso Exp $";
#endif

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <unistd.h>
#include <stdlib.h>
#include <ctype.h>

#include <gssapi/gssapi.h>
#include <gssapi/gssapi_ext.h>
#include <gss-misc.h>

#include <string.h>

void usage()
```

例 A–2 gss-server.cプログラム例の完全なコードリスト (続き)

```
{
     fprintf(stderr, "Usage: gss-server [-port port] [-verbose]\n");
     fprintf(stderr, "       [-inetd] [-logfile file] [service_name]\n");
     exit(1);
}

FILE *log;

int verbose = 0;

/*
 * Function: server_acquire_creds
 *
 * Purpose: imports a service name and acquires credentials for it
 *
 * Arguments:
 *
 *      service_name    (r) the ASCII service name
 *      server_creds    (w) the GSS-API service credentials
 *
 * Returns: 0 on success, -1 on failure
 *
 * Effects:
 *
 * The service name is imported with gss_import_name, and service
 * credentials are acquired with gss_acquire_cred.  If either operation
 * fails, an error message is displayed and -1 is returned; otherwise,
 * 0 is returned.
 */
int server_acquire_creds(service_name, server_creds)
     char *service_name;
     gss_cred_id_t *server_creds;
{
     gss_buffer_desc name_buf;
     gss_name_t server_name;
     OM_uint32 maj_stat, min_stat;

     name_buf.value = service_name;
     name_buf.length = strlen(name_buf.value) + 1;
     maj_stat = gss_import_name(&min_stat, &name_buf,
         (gss_OID) GSS_C_NT_HOSTBASED_SERVICE, &server_name);
     if (maj_stat != GSS_S_COMPLETE) {
         display_status("importing name", maj_stat, min_stat);
         return -1;
     }

     maj_stat = gss_acquire_cred(&min_stat, server_name, 0,
                                 GSS_C_NULL_OID_SET, GSS_C_ACCEPT,
                                 server_creds, NULL, NULL);
     if (maj_stat != GSS_S_COMPLETE) {
          display_status("acquiring credentials", maj_stat, min_stat);
          return -1;
     }

     (void) gss_release_name(&min_stat, &server_name);
```

例 A–2 gss-server.c プログラム例の完全なコードリスト (続き)

```
     return 0;
}

/*
 * Function: server_establish_context
 *
 * Purpose: establishes a GSS-API context as a specified service with
 * an incoming client, and returns the context handle and associated
 * client name
 *
 * Arguments:
 *
 *      s               (r) an established TCP connection to the client
 *      service_creds   (r) server credentials, from gss_acquire_cred
 *      context         (w) the established GSS-API context
 *      client_name     (w) the client's ASCII name
 *
 * Returns: 0 on success, -1 on failure
 *
 * Effects:
 *
 * Any valid client request is accepted.  If a context is established,
 * its handle is returned in context and the client name is returned
 * in client_name and 0 is returned.  If unsuccessful, an error
 * message is displayed and -1 is returned.
 */
int server_establish_context(s, server_creds, context, client_name, \
     ret_flags)

     int s;
     gss_cred_id_t server_creds;
     gss_ctx_id_t *context;
     gss_buffer_t client_name;
     OM_uint32 *ret_flags;
{
     gss_buffer_desc send_tok, recv_tok;
     gss_name_t client;
     gss_OID doid;
     OM_uint32 maj_stat, min_stat, acc_sec_min_stat;
     gss_buffer_desc    oid_name;

     *context = GSS_C_NO_CONTEXT;

     do {
          if (recv_token(s, &recv_tok) < 0)
               return -1;

          if (verbose && log) {
              fprintf(log, "Received token (size=%d): \n", recv_tok.length);
              print_token(&recv_tok);
          }

          maj_stat =
               gss_accept_sec_context(&acc_sec_min_stat,
                                      context,
                                      server_creds,
```

例 A–2 gss-server.c プログラム例の完全なコードリスト (続き)

```
                                    &recv_tok,
                                    GSS_C_NO_CHANNEL_BINDINGS,
                                    &client,
                                    &doid,
                                    &send_tok,
                                    ret_flags,
                                    NULL,     /* ignore time_rec */
                                    NULL);    /* ignore del_cred_handle */

        (void) gss_release_buffer(&min_stat, &recv_tok);

        if (send_tok.length != 0) {
            if (verbose && log) {
                fprintf(log,
                        "Sending accept_sec_context token (size=%d):\n",
                        send_tok.length);
                print_token(&send_tok);
            }
            if (send_token(s, &send_tok) < 0) {
                fprintf(log, "failure sending token\n");
                return -1;
            }

            (void) gss_release_buffer(&min_stat, &send_tok);
        }
        if (maj_stat!=GSS_S_COMPLETE && maj_stat!=GSS_S_CONTINUE_NEEDED) {
            display_status("accepting context", maj_stat,
                           acc_sec_min_stat);
            if (*context == GSS_C_NO_CONTEXT)
                    gss_delete_sec_context(&min_stat, context,
                                           GSS_C_NO_BUFFER);
            return -1;
        }

        if (verbose && log) {
            if (maj_stat == GSS_S_CONTINUE_NEEDED)
                fprintf(log, "continue needed...\n");
            else
                fprintf(log, "\n");
            fflush(log);
        }
    } while (maj_stat == GSS_S_CONTINUE_NEEDED);

    /* display the flags */
    display_ctx_flags(*ret_flags);

    if (verbose && log) {
        maj_stat = gss_oid_to_str(&min_stat, doid, &oid_name);
        if (maj_stat != GSS_S_COMPLETE) {
            display_status("converting oid->string", maj_stat, min_stat);
            return -1;
        }
        fprintf(log, "Accepted connection using mechanism OID %.*s.\n",
                (int) oid_name.length, (char *) oid_name.value);
        (void) gss_release_buffer(&min_stat, &oid_name);
    }
```

例 A–2 gss-server.c プログラム例の完全なコードリスト (続き)

```
     maj_stat = gss_display_name(&min_stat, client, client_name, &doid);
     if (maj_stat != GSS_S_COMPLETE) {
          display_status("displaying name", maj_stat, min_stat);
          return -1;
     }
     maj_stat = gss_release_name(&min_stat, &client);
     if (maj_stat != GSS_S_COMPLETE) {
          display_status("releasing name", maj_stat, min_stat);
          return -1;
     }
     return 0;
}

/*
 * Function: create_socket
 *
 * Purpose: Opens a listening TCP socket.
 *
 * Arguments:
 *
 *      port            (r) the port number on which to listen
 *
 * Returns: the listening socket file descriptor, or -1 on failure
 *
 * Effects:
 *
 * A listening socket on the specified port is created and returned.
 * On error, an error message is displayed and -1 is returned.
 */
int create_socket(port)
     u_short port;
{
     struct sockaddr_in saddr;
     int s;
     int on = 1;

     saddr.sin_family = AF_INET;
     saddr.sin_port = htons(port);
     saddr.sin_addr.s_addr = INADDR_ANY;

     if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
          perror("creating socket");
          return -1;
     }
     /* Let the socket be reused right away */
     (void) setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *)&on,
          sizeof(on));
     if (bind(s, (struct sockaddr *) &saddr, sizeof(saddr)) < 0) {
          perror("binding socket");
          (void) close(s);
          return -1;
     }
     if (listen(s, 5) < 0) {
          perror("listening on socket");
          (void) close(s);
```

例 A–2 gss-server.cプログラム例の完全なコードリスト　(続き)

```
          return -1;
     }
     return s;
}

static float timeval_subtract(tv1, tv2)
        struct timeval *tv1, *tv2;
{
        return ((tv1->tv_sec - tv2->tv_sec) +
                ((float) (tv1->tv_usec - tv2->tv_usec)) / 1000000);
}

/*
 * Yes, yes, this isn't the best place for doing this test.
 * DO NOT REMOVE THIS UNTIL A BETTER TEST HAS BEEN WRITTEN, THOUGH.
 *                                        -TYT
 */
int test_import_export_context(context)
        gss_ctx_id_t *context;
{
        OM_uint32       min_stat, maj_stat;
        gss_buffer_desc context_token, copied_token;
        struct timeval tm1, tm2;

        /*
         * Attempt to save and then restore the context.
         */
        gettimeofday(&tm1, (struct timezone *)0);
        maj_stat = gss_export_sec_context(&min_stat, context, \
            &context_token);
        if (maj_stat != GSS_S_COMPLETE) {
                display_status("exporting context", maj_stat, min_stat);
                return 1;
        }
        gettimeofday(&tm2, (struct timezone *)0);
        if (verbose && log)
                fprintf(log, "Exported context: %d bytes, %7.4f seconds\n",
                        context_token.length, timeval_subtract(&tm2, &tm1));
        copied_token.length = context_token.length;
        copied_token.value = malloc(context_token.length);
        if (copied_token.value == 0) {
            fprintf(log, "Couldn't allocate memory to copy context \
                token.\n");
            return 1;
        }
        memcpy(copied_token.value, context_token.value, \
            copied_token.length);
        maj_stat = gss_import_sec_context(&min_stat, &copied_token, \
            context);
        if (maj_stat != GSS_S_COMPLETE) {
                display_status("importing context", maj_stat, min_stat);
                return 1;
        }
        free(copied_token.value);
        gettimeofday(&tm1, (struct timezone *)0);
        if (verbose && log)
```

例 A–2 gss-server.c プログラム例の完全なコードリスト (続き)

```
                fprintf(log, "Importing context: %7.4f seconds\n",
                        timeval_subtract(&tm1, &tm2));
        (void) gss_release_buffer(&min_stat, &context_token);
        return 0;
}

/*
 * Function: sign_server
 *
 * Purpose: Performs the "sign" service.
 *
 * Arguments:
 *
 *      s               (r) a TCP socket on which a connection has been
 *                      accept()ed
 *      service_name    (r) the ASCII name of the GSS-API service to
 *                      establish a context as
 *
 * Returns: -1 on error
 *
 * Effects:
 *
 * sign_server establishes a context, and performs a single sign request.
 *
 * A sign request is a single GSS-API sealed token.  The token is
 * unsealed and a signature block, produced with gss_sign, is returned
 * to the sender.  The context is then destroyed and the connection
 * closed.
 *
 * If any error occurs, -1 is returned.
 */
int sign_server(s, server_creds)
     int s;
     gss_cred_id_t server_creds;
{
     gss_buffer_desc client_name, xmit_buf, msg_buf;
     gss_ctx_id_t context;
     OM_uint32 maj_stat, min_stat;
     int i, conf_state, ret_flags;
     char       *cp;

     /* Establish a context with the client */
     if (server_establish_context(s, server_creds, &context,
                                  &client_name, &ret_flags) < 0)
        return(-1);

     printf("Accepted connection: \"%.*s\"\n",
            (int) client_name.length, (char *) client_name.value);
     (void) gss_release_buffer(&min_stat, &client_name);

     for (i=0; i < 3; i++)
             if (test_import_export_context(&context))
                     return -1;

     /* Receive the sealed message token */
     if (recv_token(s, &xmit_buf) < 0)
```

例 A–2 gss-server.c プログラム例の完全なコードリスト　(続き)

```
        return(-1);

     if (verbose && log) {
        fprintf(log, "Sealed message token:\n");
        print_token(&xmit_buf);
     }

     maj_stat = gss_unwrap(&min_stat, context, &xmit_buf, &msg_buf,
                           &conf_state, (gss_qop_t *) NULL);
     if (maj_stat != GSS_S_COMPLETE) {
        display_status("unsealing message", maj_stat, min_stat);
        return(-1);
     } else if (! conf_state) {
        fprintf(stderr, "Warning!  Message not encrypted.\n");
     }

     (void) gss_release_buffer(&min_stat, &xmit_buf);

     fprintf(log, "Received message: ");
     cp = msg_buf.value;
     if ((isprint(cp[0]) || isspace(cp[0])) &&
         (isprint(cp[1]) || isspace(cp[1]))) {
        fprintf(log, "\"%.*s\"\n", msg_buf.length, msg_buf.value);
     } else {
        printf("\n");
        print_token(&msg_buf);
     }

     /* Produce a signature block for the message */
     maj_stat = gss_get_mic(&min_stat, context, GSS_C_QOP_DEFAULT,
                            &msg_buf, &xmit_buf);
     if (maj_stat != GSS_S_COMPLETE) {
        display_status("signing message", maj_stat, min_stat);
        return(-1);
     }

     (void) gss_release_buffer(&min_stat, &msg_buf);

     /* Send the signature block to the client */
     if (send_token(s, &xmit_buf) < 0)
        return(-1);

     (void) gss_release_buffer(&min_stat, &xmit_buf);

     /* Delete context */
     maj_stat = gss_delete_sec_context(&min_stat, &context, NULL);
     if (maj_stat != GSS_S_COMPLETE) {
        display_status("deleting context", maj_stat, min_stat);
        return(-1);
     }

     fflush(log);

     return(0);
}
```

例 A–2 gss-server.c プログラム例の完全なコードリスト (続き)

```
int
main(argc, argv)
     int argc;
     char **argv;
{
     char *service_name;
     gss_cred_id_t server_creds;
     OM_uint32 min_stat;
     u_short port = 4444;
     int s;
     int once = 0;
     int do_inetd = 0;

     log = stdout;
     display_file = stdout;
     argc--; argv++;
     while (argc) {
          if (strcmp(*argv, "-port") == 0) {
               argc--; argv++;
               if (!argc) usage();
               port = atoi(*argv);
          } else if (strcmp(*argv, "-verbose") == 0) {
              verbose = 1;
          } else if (strcmp(*argv, "-once") == 0) {
              once = 1;
          } else if (strcmp(*argv, "-inetd") == 0) {
              do_inetd = 1;
          } else if (strcmp(*argv, "-logfile") == 0) {
              argc--; argv++;
              if (!argc) usage();
              log = fopen(*argv, "a");
              display_file = log;
              if (!log) {
                  perror(*argv);
                  exit(1);
              }
          } else
               break;
          argc--; argv++;
     }
     if (argc != 1)
          usage();

     if ((*argv)[0] == '-')
          usage();

     service_name = *argv;

     if (server_acquire_creds(service_name, &server_creds) < 0)
         return -1;

     if (do_inetd) {
         close(1);
         close(2);

         sign_server(0, server_creds);
```

例 A–2 gss-server.c プログラム例の完全なコードリスト (続き)

```
        close(0);
    } else {
        int stmp;

        if ((stmp = create_socket(port)) >= 0) {
            do {
                /* Accept a TCP connection */
                if ((s = accept(stmp, NULL, 0)) < 0) {
                    perror("accepting connection");
                    continue;
                }
                /* this return value is not checked, because there's
                   not really anything to do if it fails */
                sign_server(s, server_creds);
                close(s);
            } while (!once);

            close(stmp);
        }
    }

    (void) gss_release_cred(&min_stat, &server_creds);

    /*NOTREACHED*/
    (void) close(s);
    return 0;
}
```

その他の GSS-API 関数例

クライアントプログラムやサーバープログラムが示されたとおりに動作するには、他の関数がいくつか必要です。これらの関数は、値を表示するために使用されます。それ以外では必要ありません。この種類の関数には、次のものがあります。

- `send_token()` – トークンとメッセージを受信側に転送します
- `recv_token()` – 送信側からトークンとメッセージを受け取ります
- `display_status()` – 最後に呼び出した GSS-API 関数から戻された状態を表示します
- `write_all()` – バッファーをファイルに書き込みます
- `read_all()` – ファイルからバッファーに読み込みます
- `display_ctx_flags()` – 現在のコンテキストについての情報を人が読める形式で表示します。たとえば、機密性または相互認証が許可されているかどうかなどです
- `print_token()` – トークンの値を出力します

次の例では、これらの関数のコードを示します。

注 – このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

例 A–3 その他の GSS-API 関数のコードリスト

```
/*
 * Copyright 1994 by OpenVision Technologies, Inc.
 *
 * Permission to use, copy, modify, distribute, and sell this software
 * and its documentation for any purpose is hereby granted without fee,
 * provided that the above copyright notice appears in all copies and
 * that both that copyright notice and this permission notice appear in
 * supporting documentation, and that the name of OpenVision not be used
 * in advertising or publicity pertaining to distribution of the software
 * without specific, written prior permission. OpenVision makes no
 * representations about the suitability of this software for any
 * purpose.  It is provided "as is" without express or implied warranty.
 *
 * OPENVISION DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
 * INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO
 * EVENT SHALL OPENVISION BE LIABLE FOR ANY SPECIAL, INDIRECT OR
 * CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
 * USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR
 * OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
 * PERFORMANCE OF THIS SOFTWARE.
 */

#if !defined(lint) && !defined(__CODECENTER__)
static char *rcsid = "$Header: /cvs/krbdev/krb5/src/appl/gss-sample/\
    gss-misc.c, v 1.15 1996/07/22 20:21:20 marc Exp $";
#endif

#include <stdio.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <errno.h>
#include <unistd.h>
#include <string.h>

#include <gssapi/gssapi.h>
#include <gssapi/gssapi_ext.h>
#include <gss-misc.h>

#include <stdlib.h>

FILE *display_file;

static void display_status_1
        (char *m, OM_uint32 code, int type);

static int write_all(int fildes, char *buf, unsigned int nbyte)
{
     int ret;
     char *ptr;
```

例 A–3 その他の GSS-API 関数のコードリスト (続き)

```
     for (ptr = buf; nbyte; ptr += ret, nbyte -= ret) {
          ret = write(fildes, ptr, nbyte);
          if (ret < 0) {
               if (errno == EINTR)
                    continue;
               return(ret);
          } else if (ret == 0) {
               return(ptr-buf);
          }
     }

     return(ptr-buf);
}

static int read_all(int fildes, char *buf, unsigned int nbyte)
{
     int ret;
     char *ptr;

     for (ptr = buf; nbyte; ptr += ret, nbyte -= ret) {
          ret = read(fildes, ptr, nbyte);
          if (ret < 0) {
               if (errno == EINTR)
                    continue;
               return(ret);
          } else if (ret == 0) {
               return(ptr-buf);
          }
     }

     return(ptr-buf);
}

/*
 * Function: send_token
 *
 * Purpose: Writes a token to a file descriptor.
 *
 * Arguments:
 *
 *      s               (r) an open file descriptor
 *      tok             (r) the token to write
 *
 * Returns: 0 on success, -1 on failure
 *
 * Effects:
 *
 * send_token writes the token length (as a network long) and then the
 * token data to the file descriptor s.  It returns 0 on success, and
 * -1 if an error occurs or if it could not write all the data.
 */
int send_token(s, tok)
     int s;
     gss_buffer_t tok;
{
     int len, ret;
```

例 A–3 その他の GSS-API 関数のコードリスト (続き)

```
     len = htonl(tok->length);

     ret = write_all(s, (char *) &len, 4);
     if (ret < 0) {
          perror("sending token length");
          return -1;
     } else if (ret != 4) {
         if (display_file)
             fprintf(display_file,
                     "sending token length: %d of %d bytes written\n",
                     ret, 4);
          return -1;
     }

     ret = write_all(s, tok->value, tok->length);
     if (ret < 0) {
          perror("sending token data");
          return -1;
     } else if (ret != tok->length) {
         if (display_file)
             fprintf(display_file,
                     "sending token data: %d of %d bytes written\n",
                     ret, tok->length);
         return -1;
     }

     return 0;
}

/*
 * Function: recv_token
 *
 * Purpose: Reads a token from a file descriptor.
 *
 * Arguments:
 *
 *      s               (r) an open file descriptor
 *      tok             (w) the read token
 *
 * Returns: 0 on success, -1 on failure
 *
 * Effects:
 *
 * recv_token reads the token length (as a network long), allocates
 * memory to hold the data, and then reads the token data from the
 * file descriptor s.  It blocks to read the length and data, if
 * necessary.  On a successful return, the token should be freed with
 * gss_release_buffer.  It returns 0 on success, and -1 if an error
 * occurs or if it could not read all the data.
 */
int recv_token(s, tok)
     int s;
     gss_buffer_t tok;
{
     int ret;
```

例 A–3 その他の GSS-API 関数のコードリスト (続き)

```
     ret = read_all(s, (char *) &tok->length, 4);
     if (ret < 0) {
          perror("reading token length");
          return -1;
     } else if (ret != 4) {
         if (display_file)
             fprintf(display_file,
                     "reading token length: %d of %d bytes read\n",
                     ret, 4);
         return -1;
     }

     tok->length = ntohl(tok->length);
     tok->value = (char *) malloc(tok->length);
     if (tok->value == NULL) {
         if (display_file)
             fprintf(display_file,
                     "Out of memory allocating token data\n");
          return -1;
     }

     ret = read_all(s, (char *) tok->value, tok->length);
     if (ret < 0) {
          perror("reading token data");
          free(tok->value);
          return -1;
     } else if (ret != tok->length) {
          fprintf(stderr, "sending token data: %d of %d bytes written\n",
                  ret, tok->length);
          free(tok->value);
          return -1;
     }

     return 0;
}

static void display_status_1(m, code, type)
     char *m;
     OM_uint32 code;
     int type;
{
     OM_uint32 maj_stat, min_stat;
     gss_buffer_desc msg;
     OM_uint32 msg_ctx;

     msg_ctx = 0;
     while (1) {
          maj_stat = gss_display_status(&min_stat, code,
                                        type, GSS_C_NULL_OID,
                                        &msg_ctx, &msg);
          if (display_file)
              fprintf(display_file, "GSS-API error %s: %s\n", m,
                      (char *)msg.value);
          (void) gss_release_buffer(&min_stat, &msg);
```

例 A–3 その他の GSS-API 関数のコードリスト (続き)

```
          if (!msg_ctx)
               break;
     }
}

/*
 * Function: display_status
 *
 * Purpose: displays GSS-API messages
 *
 * Arguments:
 *
 *      msg             a string to be displayed with the message
 *      maj_stat        the GSS-API major status code
 *      min_stat        the GSS-API minor status code
 *
 * Effects:
 *
 * The GSS-API messages associated with maj_stat and min_stat are
 * displayed on stderr, each preceded by "GSS-API error <msg>: " and
 * followed by a newline.
 */
void display_status(msg, maj_stat, min_stat)
     char *msg;
     OM_uint32 maj_stat;
     OM_uint32 min_stat;
{
     display_status_1(msg, maj_stat, GSS_C_GSS_CODE);
     display_status_1(msg, min_stat, GSS_C_MECH_CODE);
}

/*
 * Function: display_ctx_flags
 *
 * Purpose: displays the flags returned by context initiation in
 *          a human-readable form
 *
 * Arguments:
 *
 *      int             ret_flags
 *
 * Effects:
 *
 * Strings corresponding to the context flags are printed on
 * stdout, preceded by "context flag: " and followed by a newline
 */

void display_ctx_flags(flags)
     OM_uint32 flags;
{
     if (flags & GSS_C_DELEG_FLAG)
          fprintf(display_file, "context flag: GSS_C_DELEG_FLAG\n");
     if (flags & GSS_C_MUTUAL_FLAG)
          fprintf(display_file, "context flag: GSS_C_MUTUAL_FLAG\n");
     if (flags & GSS_C_REPLAY_FLAG)
          fprintf(display_file, "context flag: GSS_C_REPLAY_FLAG\n");
```

例 A–3 その他の GSS-API 関数のコードリスト (続き)

```
     if (flags & GSS_C_SEQUENCE_FLAG)
          fprintf(display_file, "context flag: GSS_C_SEQUENCE_FLAG\n");
     if (flags & GSS_C_CONF_FLAG )
          fprintf(display_file, "context flag: GSS_C_CONF_FLAG \n");
     if (flags & GSS_C_INTEG_FLAG )
          fprintf(display_file, "context flag: GSS_C_INTEG_FLAG \n");
}

void print_token(tok)
     gss_buffer_t tok;
{
    int i;
    unsigned char *p = tok->value;

    if (!display_file)
        return;
    for (i=0; i < tok->length; i++, p++) {
        fprintf(display_file, "%02x ", *p);
        if ((i % 16) == 15) {
            fprintf(display_file, "\n");
        }
    }
    fprintf(display_file, "\n");
    fflush(display_file);
}
```

◆ ◆ ◆ 付 録 B

GSS-API リファレンス

この付録は、次のような節から構成されています。

- 227 ページの「GSS-API 関数」では、GSS-API 関数の表を示します。
- 230 ページの「GSS-API 状態コード」では、GSS-API 関数が戻す状態コードについて説明し、状態コードのリストを示します。
- 234 ページの「GSS-API データ型と値」では、GSS-API で使用されるさまざまなデータ型について説明します。
- 237 ページの「GSS-API の実装に固有な機能」では、GSS-API の Sun の実装に固有な機能について説明します。
- 240 ページの「Kerberos v5 状態コード」では、Kerberos v5 機構で戻ることがある状態コードのリストを示します。

これ以外の GSS-API 定義については、ファイル `gssapi.h` を参照してください。

GSS-API 関数

Solaris ソフトウェアでは、次の GSS-API 関数を実装します。各関数の詳細は、それぞれのマニュアルページを参照してください。また、229 ページの「旧バージョンの GSS-API 関数」も参照してください。

`gss_acquire_cred()`	すでに存在している資格の GSS-API 資格ハンドルを取得することにより、大域的な ID を想定します
`gss_add_cred()`	資格を増分的に作成します
`gss_inquire_cred()`	資格に関する情報を取得します
`gss_inquire_cred_by_mech()`	資格に関する機構ごとの情報を取得します
`gss_release_cred()`	資格ハンドルを破棄します

`gss_init_sec_context()`	ピアとなるアプリケーションでセキュリティーコンテキストを起動します
`gss_accept_sec_context()`	ピアとなるアプリケーションが起動したセキュリティーコンテキストを受け入れます
`gss_delete_sec_context()`	セキュリティーコンテキストを破棄します
`gss_process_context_token()`	ピアとなるアプリケーションからのセキュリティーコンテキストでトークンを処理します
`gss_context_time()`	コンテキストが有効である時間を決定します
`gss_inquire_context()`	セキュリティーコンテキストに関する情報を取得します
`gss_wrap_size_limit()`	`gss_wrap()` をコンテキストで実行するためにトークンのサイズの制限を決定します
`gss_export_sec_context()`	セキュリティーコンテキストを別のプロセスに転送します
`gss_import_sec_context()`	転送されたコンテキストをインポートします
`gss_get_mic()`	メッセージの暗号化メッセージ整合性コード (MIC) を計算します
`gss_verify_mic()`	MIC とメッセージを照合して、受信したメッセージの整合性を検証します
`gss_wrap()`	MIC をメッセージに添付し、メッセージの内容を暗号化します (後者は省略可能)
`gss_unwrap()`	添付された MIC でメッセージを検証します。必要に応じて、メッセージの内容を復号化します
`gss_import_name()`	連続する文字列名を内部形式名に変換します
`gss_display_name()`	内部形式名をテキストに変換します
`gss_compare_name()`	2 つの内部形式名を比較します
`gss_release_name()`	内部形式名を破棄します
`gss_inquire_names_for_mech()`	指定した機構がサポートする名前型のリストを表示します
`gss_inquire_mechs_for_name()`	指定した名前型をサポートする機構のリストを表示します
`gss_canonicalize_name()`	内部名を機構名 (MN) に変換します
`gss_export_name()`	MN をエクスポート形式に変換します

`gss_duplicate_name()`	内部名のコピーを作成します
`gss_add_oid_set_member()`	オブジェクト識別子を集合に追加します
`gss_display_status()`	GSS-API 状態コードをテキストに変換します
`gss_indicate_mechs()`	使用できる実際の認証機構を決定します
`gss_release_buffer()`	バッファーを破棄します
`gss_release_oid_set()`	オブジェクト識別子の集合を破棄します
`gss_create_empty_oid_set()`	オブジェクト識別子の空の集合を作成します
`gss_test_oid_set_member()`	オブジェクト識別子が集合のメンバーであるかどうかを確認します

旧バージョンの GSS-API 関数

この節では、旧バージョンの GSS-API 関数について説明します。

OID を処理する関数

GSS-API の Sun の実装では、便宜上および下位互換性のために、次の関数が提供されています。しかし、これらの関数は GSS-API の Sun 以外の実装ではサポートされていない場合があります。

- `gss_delete_oid()`
- `gss_oid_to_str()`
- `gss_str_to_oid()`

機構名は文字列から OID に変換されますが、可能な限り、GSS-API が提供するデフォルトの機構を使用してください。

名前が変更された関数

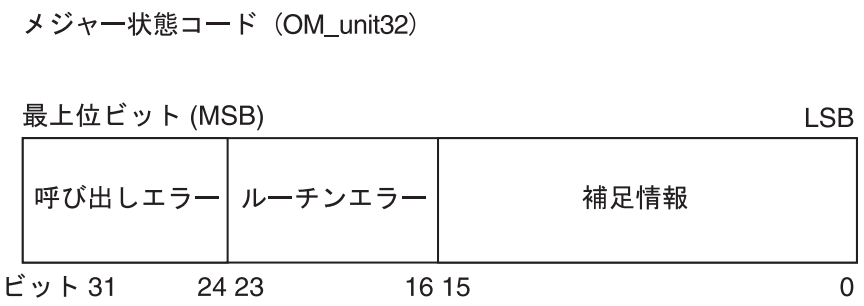
次の関数は新しい関数に差し替えられました。どの場合も、新しい関数は古い関数と機能的に同等です。古い関数もサポートされていますが、可能な限り、新しい関数に置き換えてください。

- `gss_sign()` は `gss_get_mic()` に置き換えられました。
- `gss_verify()` は `gss_verify_mic()` に置き換えられました。
- `gss_seal()` は `gss_wrap()` に置き換えられました。
- `gss_unseal()` は `gss_unwrap()` に置き換えられました。

GSS-API 状態コード

メジャー状態コードは、次の図に示すように、OM_uint32 に符号化されます。

図 B–1 メジャー状態の符号化



GSS-API ルーチンが上位 16 ビットに 0 以外の値が入った GSS 状態コードを戻す場合、その呼び出しが失敗したことを示します。呼び出しエラーフィールドが 0 以外の場合、アプリケーションのルーチンの呼び出しにエラーがあったことを示します。表 B–1 に、「呼び出しエラー」のリストを示します。ルーチンエラーフィールドが 0 以外の場合、ルーチン固有のエラーのためにルーチンが失敗したことを示します。表 B–2 に、「ルーチン固有のエラー」のリストを示します。上位 16 ビットが失敗または成功のどちらを示すかにかかわらず、状態コードの補足情報フィールドのビットを設定できます。表 B–3 に、個々のビットの意味を示します。

GSS-API メジャー状態コードの値

次の表に、GSS-API が戻す呼び出しエラーのリストを示します。これらのエラーは、特定の言語バインディング (この場合は C) に固有です。

表 B–1 GSS-API の呼び出しエラー

エラー	フィールドの値	意味
GSS_S_CALL_INACCESSIBLE_READ	1	要求された入力パラメータを読み取れません
GSS_S_CALL_INACCESSIBLE_WRITE	2	要求された出力パラメータに書き込めません

表B–1 GSS-APIの呼び出しエラー (続き)

エラー	フィールドの値	意味
GSS_S_CALL_BAD_STRUCTURE	3	パラメータの形式が間違っています

次の表に、GSS-APIルーチンエラー(GSS-API関数が戻す一般的なエラー)のリストを示します。

表B–2 GSS-APIルーチンエラー

エラー	フィールドの値	意味
GSS_S_BAD_MECH	1	要求された機構がサポートされていません。
GSS_S_BAD_NAME	2	提供された名前が無効です。
GSS_S_BAD_NAMETYPE	3	提供された名前型がサポートされていません。
GSS_S_BAD_BINDINGS	4	提供されたチャネルバインディングが間違っています。
GSS_S_BAD_STATUS	5	提供された状態コードが無効です。
GSS_S_BAD_MIC, GSS_S_BAD_SIG	6	トークンが持っているMICが無効です。
GSS_S_NO_CRED	7	資格を使用またはアクセスできません。あるいは、資格が提供されていません。
GSS_S_NO_CONTEXT	8	コンテキストがまったく確立されていません。
GSS_S_DEFECTIVE_TOKEN	9	トークンが無効です。
GSS_S_DEFECTIVE_CREDENTIAL	10	資格が無効です。
GSS_S_CREDENTIALS_EXPIRED	11	参照された資格の有効期間が終了しています。
GSS_S_CONTEXT_EXPIRED	12	コンテキストの有効期間が終了しています。
GSS_S_FAILURE	13	その他のエラー。実際の機構によって、特定のGSS-API状態コードが定義されていないエラーが検出されました。この場合、機構に固有の状態コード(マイナー状態コード)にエラーの詳細が示されます。

表 B–2 GSS-API ルーチンエラー (続き)

エラー	フィールドの値	意味
GSS_S_BAD_QOP	14	要求された保護品質を提供できません。
GSS_S_UNAUTHORIZED	15	当該操作はローカルのセキュリティーポリシーによって禁止されています。
GSS_S_UNAVAILABLE	16	当該操作またはオプションは使用できません。
GSS_S_DUPLICATE_ELEMENT	17	要求された資格要素はすでに存在しています。
GSS_S_NAME_NOT_MN	18	提供された名前が機構名 (MN) ではありません。

GSS_S_COMPLETE という名前は、値が 0 で、API エラーまたは補足情報ビットのどちらも存在しないことを示します。

次の表に、GSS-API 関数が戻す補足情報の値のリストを示します。

表 B–3 GSS-API 補足情報コード

コード	ビット番号	意味
GSS_S_CONTINUE_NEEDED	0 (LSB)	gss_init_sec_context() または gss_accept_sec_context() だけが戻します。関数を完了させるには、もう一度ルーチンを呼び出す必要があることを示します。
GSS_S_DUPLICATE_TOKEN	1	トークンは以前のトークンの複製です。
GSS_S_OLD_TOKEN	2	トークンの有効期間が終了しています。
GSS_S_UNSEQ_TOKEN	3	後方にあるトークンをすでに処理しています。
GSS_S_GAP_TOKEN	4	期待していたメッセージ毎トークンを受信していません。

状態コードの詳細は、71 ページの「GSS-API 状態コード」を参照してください。

状態コードの表示

gss_display_status() 関数は、GSS-API 状態コードをテキスト形式に変換します。この形式を使用すると、コードをユーザーに表示したり、テキストログに格納したりできます。関数の中には複数の状態を戻すものもありますが、gss_display_status()

関数は一度に 1 つの状態コードしか表示できません。このため、gss_display_status() をループの一部として呼び出す必要があります。gss_display_status() が 0 以外の状態コードを示すときは、関数は別の状態コードを取得できます。

例 B–1 gss_display_status() による状態コードの表示

```
OM_uint32 message_context;
OM_uint32 status_code;
OM_uint32 maj_status;
OM_uint32 min_status;
gss_buffer_desc status_string;

...

message_context = 0;

do {

     maj_status = gss_display_status(
               &min_status,
               status_code,
               GSS_C_GSS_CODE,
               GSS_C_NO_OID,
               &message_context,
               &status_string);

     fprintf(stderr, "%.*s\n", \
               (int)status_string.length, \
               (char *)status_string.value);

     gss_release_buffer(&min_status, &status_string,);

} while (message_context != 0);
```

状態コードのマクロ

マクロ GSS_CALLING_ERROR()、GSS_ROUTINE_ERROR ()、および GSS_SUPPLEMENTARY_INFO() は GSS 状態コードを受け取ります。これらのマクロは、関係のあるフィールド以外の情報をすべて削除します。たとえば、GSS_ROUTINE_ERROR() を状態コードに適用すると、呼び出しエラーフィールドと補足情報フィールドは削除されます。この操作では、ルーチンエラーフィールドのみが残ります。このようなマクロが提供する値は、適切な型の GSS_S_*xxx* シンボルと直接比較できます。また、マクロ GSS_ERROR() は、状態コードが呼び出しエラーまたはルーチンエラーを示す場合は 0 以外の値を戻し、そうでない場合は 0 を戻します。GSS-API で定義されるすべてのマクロは引数を 1 つだけしか受け取りません。

GSS-API データ型と値

この節では、さまざまなタイプの GSS-API データ型と値について説明します。gss_cred_id_t または gss_name_t などのいくつかのデータ型はユーザーに不透明です。これらのデータ型については説明する必要がありません。この節では、次の項目について説明します。

- 234 ページの「基本 GSS-API データ型」 – OM_uint32、gss_buffer_desc、gss_OID_desc、gss_OID_set_desc_struct、gss_channel_bindings の各データ型の定義を示します。
- 235 ページの「名前型」 – 名前を指定するときに GSS-API が認識する各種の名前の形式を示します。
- 236 ページの「チャネルバインディングのアドレス型」 – gss_channel_bindings_t 構造体の *initiator_addrtype* および *acceptor_addrtype* の各フィールドで使用できるさまざまな値を示します。

基本 GSS-API データ型

ここでは、GSS-API で使用されるデータ型について説明します。

OM_uint32

OM_uint32 はプラットフォームに依存しない 32 ビットの符号なし整数です。

gss_buffer_desc

gss_buffer_t ポインタを含む gss_buffer_desc の定義は、次の書式で記述されます。

```
typedef struct gss_buffer_desc_struct {
        size_t length;
        void *value;
} gss_buffer_desc, *gss_buffer_t;
```

gss_OID_desc

gss_OID ポインタを含む gss_OID_desc の定義は、次の書式で記述されます。

```
typedef struct gss_OID_desc_struct {
        OM_uint32 length;
        void*elements;
} gss_OID_desc, *gss_OID;
```

gss_OID_set_desc

gss_OID_set ポインタを含む gss_OID_set_desc の定義は、次の書式で記述されます。

```
typedef struct gss_OID_set_desc_struct  {
        size_t  count;
        gss_OID elements;
} gss_OID_set_desc, *gss_OID_set;
```

gss_channel_bindings_struct

gss_channel_bindings_struct 構造体と gss_channel_bindings_t ポインタの定義は、次の書式で記述されます。

```
typedef struct gss_channel_bindings_struct {
        OM_uint32 initiator_addrtype;
        gss_buffer_desc initiator_address;
        OM_uint32 acceptor_addrtype;
        gss_buffer_desc acceptor_address;
        gss_buffer_desc application_data;
} *gss_channel_bindings_t;
```

名前型

名前型は、関連する名前の形式を示しています。名前と名前型の詳細は、63 ページの「GSS-API における名前」および 70 ページの「GSS-API の OID」を参照してください。GSS-API は、次の表に示す gss_OID 名前型をサポートしています。

GSS_C_NO_NAME
: シンボリック名 GSS_C_NO_NAME は、名前の転送で提供される値がないことを示すパラメータ値として推奨されます。

GSS_C_NO_OID
: 実際のオブジェクト識別子ではなく、NULL の入力値に相当します。この値を指定した場合、関連する名前が機構に固有なデフォルトの印刷可能な構文に基づいて解釈されることを示します。

GSS_C_NT_ANONYMOUS
: 匿名を確認する方法。この値と比較することによって、名前が匿名の主体を参照するかどうかを機構に依存しない方法で確認できます。

GSS_C_NT_EXPORT_NAME
: gss_export_name() 関数でエクスポートされた名前。

GSS_C_NT_HOSTBASED_SERVICE
: ホストコンピュータに関連付けられたサービスを表します。この名前型は、サービス (service) とホスト名 (hostname) の 2 つの要素からなり、service@hostname の書式で記述されます。

GSS_C_NT_MACHINE_UID_NAME
: ローカルシステム上のユーザーの、数値によるユーザー識別子を示します。この値の解釈方法は OS に固有です。gss_import_name() 関数はこの UID をユーザー名に解釈処理し、ユーザー名形式として扱います。

GSS_C_NT_STRING_STRING_UID_NAME
: ローカルシステム上のユーザーの数値によるユーザー識別子を表す数字文字列を示します。この値の解釈方法は OS に固有です。この名前型はマシン UID 形式に似ていますが、バッファーにはユーザー ID を表す文字が入っています。

GSS_C_NT_USER_NAME
: ローカルシステム上の指定されたユーザー。この値の解釈方法は OS に固有です。この値は *username* の書式で記述されます。

チャネルバインディングのアドレス型

次の表に、`gss_channel_bindings_struct` 構造体の *initiator_addrtype* フィールドと *acceptor_addrtype* フィールドで使用できる値を示します。この 2 つのフィールドは、名前を受け取ることができる形式 (ARPAnet IMP アドレスや AppleTalk アドレスなど) を示します。チャネルバインディングについては、83 ページの「GSS-API におけるチャネルバインディングの使用」を参照してください。

表 B–4 チャネルバインディングのアドレス型

フィールド	値 (10 進数)	アドレス型
GSS_C_AF_UNSPEC	0	未定のアドレス型
GSS_C_AF_LOCAL	1	ホスト - ローカル
GSS_C_AF_INET	2	インターネットアドレス型 (IP など)
GSS_C_AF_IMPLINK	3	ARPAnet IMP
GSS_C_AF_PUP	4	pup プロトコル (BSP など)
GSS_C_AF_CHAOS	5	MIT CHAOS プロトコル
GSS_C_AF_NS	6	XEROX NS
GSS_C_AF_NBS	7	nbs
GSS_C_AF_ECMA	8	ECMA
GSS_C_AF_DATAKIT	9	データキットプロトコル
GSS_C_AF_CCITT	10	CCITT
GSS_C_AF_SNA	11	IBM SNA
GSS_C_AF_DECnet	12	DECnet
GSS_C_AF_DLI	13	ダイレクトデータリンクインタフェース
GSS_C_AF_LAT	14	LAT
GSS_C_AF_HYLINK	15	NSC ハイパーチャネル

表 B–4 チャネルバインディングのアドレス型 (続き)

フィールド	値 (10 進数)	アドレス型
GSS_C_AF_APPLETALK	16	AppleTalk
GSS_C_AF_BSC	17	BISYNC
GSS_C_AF_DSS	18	分散システムサービス
GSS_C_AF_OSI	19	OSI TP4
GSS_C_AF_X25	21	X.25
GSS_C_AF_NULLADDR	255	アドレスは指定されていません

GSS-API の実装に固有な機能

GSS-API の実装によっては、いくつかの動作が若干異なる場合もあります。ほとんどの場合、実装による違いはプログラムに最小限の影響しか与えません。どのような場合でも、実装に固有な動作 (Sun の実装も含む) に依存しなければ、移植性を最大限にすることができます。

Sun 固有の関数

Sun の実装には、カスタマイズされた GSS-API 関数はありません。

人が読める名前についての構文

GSS-API の実装によっては、名前の出力可能な形式についての構文が異なる場合があります。移植性を最大限にする場合は、アプリケーションで、人が読める (つまり出力可能な) 形式を使用する名前を比較するべきではありません。そのかわり、`gss_compare_name()` を使用して内部形式名がほかの名前と一致するかどうかを確認してください。

Sun の実装の `gss_display_name()` では名前を次のように表示します。*input_name* 引数がユーザー主体を指す場合、`gss_display_name()` は `user_principal@realm` を *output_name_buffer* として、`gss_OID` 値を *output_name_type* として戻します。Kerberos v5 が実際の機構である場合、`gss_OID` は `1.2.840.11354.1.2.2` になります。

`gss_import_name()` が `GSS_C_NO_OID` 名前型で作成した名前を `gss_display_name()` が受け取った場合、`gss_display_name()` は `GSS_C_NO_OID` を *output_name_type* パラメータで戻します。

匿名の形式

gss_display_name() 関数は、匿名の GSS-API 主体を示すとき、文字列 <anonymous> を出力します。この名前に関連する名前型 OID は GSS_C_NT_ANONYMOUS です。Sun の実装で有効な印刷可能な名前の中では、これ以外にかぎかっこ (<>) で囲まれているものはありません。

選択されたデータ型の実装

gss_cred_t、gss_ctx_id_t、gss_name_t の各データ型はポインタとして実装されています (一部の実装では算術型として指定される場合もある)。

コンテキストの削除と格納されたデータの解放

コンテキストの確立に失敗すると、Sun の実装では部分的に構築されたコンテキストを自動的に削除しません。したがって、アプリケーションでこの事態を処理する、つまり、gss_delete_sec_context() でコンテキストを削除する必要があります。

Sun の実装では、格納されたデータ (内部名など) を、メモリー管理を通じて自動的に解放します。しかし、データ要素が必要でなくなったときには、アプリケーションで適切な関数 (gss_release_name() など) を呼び出す必要があります。

チャネルバインディング情報の保護

チャネルバインディングをサポートしているかどうかは、機構によって異なります。Diffie-Hellman 機構と Kerberos v5 機構はどちらもチャネルバインディングをサポートしています。

開発者は、チャネルバインディングデータには機密保護が施されていないものとしてください。Kerberos v5 機構には機密保護が用意されていますが、 Diffie-Hellman 機構ではチャネルバインディングデータの機密性は保持されません。

コンテキストのエクスポートとプロセス間トークン

Sun の実装では、同じコンテキストに対する複数のインポートの試みを検出および拒否します。

サポートされる資格の型

GSS-API の Sun の実装では、`gss_acquire_cred()` による、`GSS_C_INITIATE`、`GSS_C_ACCEPT`、および `GSS_C_BOTH` の各資格の獲得をサポートしています。

資格の有効期間

GSS-API の Sun の実装では、資格の有効期間の設定をサポートします。したがって、プログラマは `gss_acquire_cred()` や `gss_add_cred()` などの関数で、資格の有効期間に関連するパラメータを使用できます。

コンテキストの有効期間

GSS-API の Sun の実装では、コンテキストの有効期間の設定をサポートします。したがって、プログラマは `gss_init_sec_context()` や `gss_inquire_context()` などの関数で、コンテキストの有効期間に関連するパラメータを使用できます。

ラップサイズの制限と **QOP** 値

GSS-API の Sun 実装では、基礎となる機構とは異なり、`gss_wrap()` で処理するメッセージの最大サイズ制限を課しません。アプリケーションは `gss_wrap_size_limit()` でメッセージの最大サイズを決定できます。

GSS-API の Sun の実装では、`gss_wrap_size_limit()` を呼び出すとき、無効な QOP 値を検出します。

minor_status パラメータの使用

GSS-API の Sun の実装では、関数が *minor_status* パラメータで戻すのは、機構に固有な情報のみです。他の実装では、戻されたマイナー状態コードの一部として実装に固有な戻り値が含まれることもあります。

Kerberos v5 状態コード

各 GSS-API 関数は 2 つの状態コードを戻します。「メジャー状態コード」と「マイナー状態コード」です。メジャー状態コードは GSS-API の動作に関連します。たとえば、セキュリティーコンテキストの有効期間が終了したあとで、アプリケーションがメッセージを転送しようとした場合、GSS-API は `GSS_S_CONTEXT_EXPIRED` というメジャー状態コードを戻します。メジャー状態コードのリストについては、230 ページの「GSS-API 状態コード」を参照してください。

マイナー状態コードを戻すのは、GSS-API の実装でサポートされる実際のセキュリティー機構です。すべての GSS-API 関数は、最初の引数として *minor_status* または *minor_stat* パラメータを受け取ります。関数が戻ったときにこのパラメータを調べることによって、アプリケーションは、関数が成功したかどうかにかかわらず、実際の機構が戻した状態を知ることができます。

次の表に、Kerberos v5 が *minor_status* 引数に戻す状態メッセージのリストを示します。GSS-API 状態コードの詳細は、71 ページの「GSS-API 状態コード」を参照してください。

Kerberos v5 で状態コード 1 として戻されるメッセージ

次の表に、Kerberos v5 で状態コード 1 として戻されるマイナー状態メッセージのリストを示します。

表 B–5 Kerberos v5 状態コード 1

マイナー状態	値	意味
KRB5KDC_ERR_NONE	-1765328384L	エラーなし
KRB5KDC_ERR_NAME_EXP	-1765328383L	データベース内のクライアントのエントリの有効期間が終了しています
KRB5KDC_ERR_SERVICE_EXP	-1765328382L	データベース内のサーバーのエントリの有効期間が終了しています
KRB5KDC_ERR_BAD_PVNO	-1765328381L	Requested protocol version not supported (要求したプロトコルバージョンはサポートされていません。)
KRB5KDC_ERR_C_OLD_MAST_KVNO	-1765328380L	クライアントの鍵が古いマスター鍵で暗号化されています

表 B–5 Kerberos v5 状態コード 1 *(続き)*

マイナー状態	値	意味
KRB5KDC_ERR_S_OLD_MAST_KVNO	-1765328379L	サーバーの鍵が古いマスター鍵で暗号化されています
KRB5KDC_ERR_C_PRINCIPAL_UNKNOWN	-1765328378L	クライアントが Kerberos データベースに見つかりません
KRB5KDC_ERR_S_PRINCIPAL_UNKNOWN	-1765328377L	サーバーが Kerberos データベースに見つかりません
KRB5KDC_ERR_PRINCIPAL_NOT_UNIQUE	-1765328376L	主体が Kerberos データベースに複数のエントリを持っています
KRB5KDC_ERR_NULL_KEY	-1765328375L	Client or server has a null key (クライアントまたはサーバーの鍵が空です。)
KRB5KDC_ERR_CANNOT_POSTDATE	-1765328374L	Ticket is ineligible for postdating (チケットには遅延処理の資格がありません。)
KRB5KDC_ERR_NEVER_VALID	-1765328373L	要求された有効期間が負であるか、短すぎます
KRB5KDC_ERR_POLICY	-1765328372L	KDC policy rejects request (KDC ポリシーは要求を拒否します。)
KRB5KDC_ERR_BADOPTION	-1765328371L	KDC can't fulfill requested option (KDC は要求したオプションを処理できません。)
KRB5KDC_ERR_ETYPE_NOSUPP	-1765328370L	KDC が暗号化型をサポートしていません
KRB5KDC_ERR_SUMTYPE_NOSUPP	-1765328369L	KDC がチェックサム型をサポートしていません
KRB5KDC_ERR_PADATA_TYPE_NOSUPP	-1765328368L	KDC は padata タイプをサポートしていません。
KRB5KDC_ERR_TRTYPE_NOSUPP	-1765328367L	KDC は transited タイプをサポートしていません。

表 B–5 Kerberos v5 状態コード 1 *(続き)*

マイナー状態	値	意味
KRB5KDC_ERR_CLIENT_REVOKED	-1765328366L	クライアントの資格が取り消されました
KRB5KDC_ERR_SERVICE_REVOKED	-1765328365L	サーバーの資格が取り消されました

Kerberos v5 で状態コード 2 として戻されるメッセージ

次の表に、Kerberos v5 で状態コード 2 として戻されるマイナー状態メッセージのリストを示します。

表 B–6 Kerberos v5 状態コード 2

マイナー状態	値	意味
KRB5KDC_ERR_TGT_REVOKED	-1765328364L	TGT が取り消されました
KRB5KDC_ERR_CLIENT_NOTYET	-1765328363L	クライアントがまだ有効ではありません。のちほど再試行してください
KRB5KDC_ERR_SERVICE_NOTYET	-1765328362L	サーバーがまだ有効ではありません。のちほど再試行してください
KRB5KDC_ERR_KEY_EXP	-1765328361L	パスワードの有効期間が終了しています
KRB5KDC_ERR_PREAUTH_FAILED	-1765328360L	事前認証が失敗しました
KRB5KDC_ERR_PREAUTH_REQUIRED	-1765328359L	追加の事前認証が要求されました
KRB5KDC_ERR_SERVER_NOMATCH	-1765328358L	要求されたサーバーとチケットが一致しません
KRB5PLACEHOLD_27 ~ KRB5PLACEHOLD_30	-1765328357L -1765328354L	KRB5 エラーコード (27 30。予約済み)
KRB5KRB_AP_ERR_BAD_INTEGRITY	-1765328353L	Decrypt integrity check failed (復号化で整合性チェックが失敗しました。)
KRB5KRB_AP_ERR_TKT_EXPIRED	-1765328352L	Ticket expired (チケットの有効期限が切れました。)

表 B–6 Kerberos v5 状態コード 2 *(続き)*

マイナー状態	値	意味
KRB5KRB_AP_ERR_TKT_NYV	-1765328351L	Ticket not yet valid (チケットはまだ有効ではありません。)
KRB5KRB_AP_ERR_REPEAT	-1765328350L	Request is a replay (要求は再送です。)
KRB5KRB_AP_ERR_NOT_US	-1765328349L	The ticket isn't for us (チケットはわれわれのものではありません。)
KRB5KRB_AP_ERR_BADMATCH	-1765328348L	チケットと認証用データが一致しません
KRB5KRB_AP_ERR_SKEW	-1765328347L	クロックスキューが大きすぎます
KRB5KRB_AP_ERR_BADADDR	-1765328346L	Incorrect net address (ネットアドレスが間違っています。)
KRB5KRB_AP_ERR_BADVERSION	-1765328345L	Protocol version mismatch (プロトコルバージョンが一致していません。)
KRB5KRB_AP_ERR_MSG_TYPE	-1765328344L	メッセージの型が無効です
KRB5KRB_AP_ERR_MODIFIED	-1765328343L	Message stream modified (メッセージストリームが変更されました。)
KRB5KRB_AP_ERR_BADORDER	-1765328342L	Message out of order (メッセージの順序が違います。)
KRB5KRB_AP_ERR_ILL_CR_TKT	-1765328341L	Illegal cross-realm ticket (レルム間のチケットが無効です。)
KRB5KRB_AP_ERR_BADKEYVER	-1765328340L	キーのバージョンが使用できません

Kerberos v5 で状態コード 3 として戻されるメッセージ

次の表に、Kerberos v5 で状態コード 3 として戻されるマイナー状態メッセージのリストを示します。

表 B–7 Kerberos v5 状態コード 3

マイナー状態	値	意味
KRB5KRB_AP_ERR_NOKEY	-1765328339L	Service key not available (サービス鍵が使用できません。)
KRB5KRB_AP_ERR_MUT_FAIL	-1765328338L	相互認証が失敗しました
KRB5KRB_AP_ERR_BADDIRECTION	-1765328337L	メッセージの方向が間違っています
KRB5KRB_AP_ERR_METHOD	-1765328336L	代替の認証方法が要求されました
KRB5KRB_AP_ERR_BADSEQ	-1765328335L	メッセージ内のシーケンス番号が間違っています
KRB5KRB_AP_ERR_INAPP_CKSUM	-1765328334L	Inappropriate type of checksum in message (メッセージのチェックサムのタイプが不適切です。)
KRB5PLACEHOLD_51 ~ KRB5PLACEHOLD_59	-1765328333L -1765328325L	KRB5 エラーコード (51 59。予約済み)
KRB5KRB_ERR_GENERIC	-1765328324L	一般的なエラー
KRB5KRB_ERR_FIELD_TOOLONG	-1765328323L	Field is too long for this implementation (この実装ではフィールドが長すぎます。)
KRB5PLACEHOLD_62 ~ KRB5PLACEHOLD_127	-1765328322L -1765328257L	KRB5 エラーコード (62 127。予約済み)
値は戻されない	-1765328256L	内部使用のみ
KRB5_LIBOS_BADLOCKFLAG	-1765328255L	Invalid flag for file lock mode (ファイルロックモードのフラグが無効です。)
KRB5_LIBOS_CANTREADPWD	-1765328254L	パスワードを読み取れません
KRB5_LIBOS_BADPWDMATCH	-1765328253L	パスワードが一致しません
KRB5_LIBOS_PWDINTR	-1765328252L	パスワードの読み取りが中断されました

表 B–7 Kerberos v5 状態コード 3 *(続き)*

マイナー状態	値	意味
KRB5_PARSE_ILLCHAR	-1765328251L	構成要素名の文字が無効です
KRB5_PARSE_MALFORMED	-1765328250L	主体の表現形式が間違っています
KRB5_CONFIG_CANTOPEN	-1765328249L	Kerberos 構成ファイル `/etc/krb5/krb5` が開けません (または、見つかりません)
KRB5_CONFIG_BADFORMAT	-1765328248L	Kerberos 構成ファイル `/etc/krb5/krb5` の形式が不適切です
KRB5_CONFIG_NOTENUFSPACE	-1765328247L	完全な情報を戻すには領域が不足しています
KRB5_BADMSGTYPE	-1765328246L	符号化用に指定したメッセージ型が無効です
KRB5_CC_BADNAME	-1765328245L	資格キャッシュ名の形式が間違っています

Kerberos v5 で状態コード 4 として戻されるメッセージ

次の表に、Kerberos v5 で状態コード 4 として戻されるマイナー状態メッセージのリストを示します。

表 B–8 Kerberos v5 状態コード 4

マイナー状態	値	意味
KRB5_CC_UNKNOWN_TYPE	-1765328244L	資格キャッシュ型が不明です
KRB5_CC_NOTFOUND	-1765328243L	一致する資格が見つかりません
KRB5_CC_END	-1765328242L	資格キャッシュの終わりに到達しました
KRB5_NO_TKT_SUPPLIED	-1765328241L	要求がチケットを提供していません

表 B–8 Kerberos v5 状態コード 4 *(続き)*

マイナー状態	値	意味
KRB5KRB_AP_WRONG_PRINC	-1765328240L	Wrong principal in request (要求した主体は正しくありません。)
KRB5KRB_AP_ERR_TKT_INVALID	-1765328239L	チケットが設定したフラグが無効です
KRB5_PRINC_NOMATCH	-1765328238L	Requested principal and ticket don't match (要求した主体とチケットは一致しません。)
KRB5_KDCREP_MODIFIED	-1765328237L	KDC reply did not match expectations (KDC 応答は予期したものと一致しませんでした。)
KRB5_KDCREP_SKEW	-1765328236L	クロックスキューが KDC 返信には大きすぎます
KRB5_IN_TKT_REALM_MISMATCH	-1765328235L	Client/server realm mismatch in initial ticket request (初期チケット要求でクライアント/サーバーレルムが一致していません。)
KRB5_PROG_ETYPE_NOSUPP	-1765328234L	プログラムが暗号化型をサポートしていません
KRB5_PROG_KEYTYPE_NOSUPP	-1765328233L	プログラムが鍵型をサポートしていません
KRB5_WRONG_ETYPE	-1765328232L	要求された暗号化型がメッセージで使用されていません
KRB5_PROG_SUMTYPE_NOSUPP	-1765328231L	プログラムがチェックサム型をサポートしていません
KRB5_REALM_UNKNOWN	-1765328230L	Cannot find KDC for requested realm (要求されたレルムの KDC が見つかりません。)
KRB5_SERVICE_UNKNOWN	-1765328229L	Kerberos サービスが不明です

表 B–8 Kerberos v5 状態コード 4 *(続き)*

マイナー状態	値	意味
KRB5_KDC_UNREACH	-1765328228L	Cannot contact any KDC for requested realm (要求されたレルムの KDC に接続できません。)
KRB5_NO_LOCALNAME	-1765328227L	主体名のローカル名が見つかりません
KRB5_MUTUAL_FAILED	-1765328226L	相互認証が失敗しました
KRB5_RC_TYPE_EXISTS	-1765328225L	リプレイのキャッシュ型がすでに登録されています
KRB5_RC_MALLOC	-1765328224L	リプレイのキャッシュコードでこれ以上のメモリーを割り当てられません
KRB5_RC_TYPE_NOTFOUND	-1765328223L	リプレイのキャッシュ型が不明です

Kerberos v5 で状態コード 5 として戻されるメッセージ

次の表に、Kerberos v5 で状態コード 5 として戻されるマイナー状態メッセージのリストを示します。

表 B–9 Kerberos v5 状態コード 5

マイナー状態	値	意味
KRB5_RC_UNKNOWN	-1765328222L	一般的な不明な RC エラー
KRB5_RC_REPLAY	-1765328221L	リプレイされたメッセージ
KRB5_RC_IO	-1765328220L	リプレイの入出力操作が失敗しました
KRB5_RC_NOIO	-1765328219L	リプレイのキャッシュ型が非揮発性記憶装置をサポートしません
KRB5_RC_PARSE	-1765328218L	リプレイのキャッシュ名の解析および形式エラー

表 B–9 Kerberos v5 状態コード 5 *(続き)*

マイナー状態	値	意味
KRB5_RC_IO_EOF	-1765328217L	リプレイのキャッシュ入出力でファイルの終わりに到達しました
KRB5_RC_IO_MALLOC	-1765328216L	リプレイのキャッシュ入出力コードでこれ以上メモリーを割り当てられません
KRB5_RC_IO_PERM	-1765328215L	Permission denied in replay cache code (再実行キャッシュコードでアクセス権がありません。)
KRB5_RC_IO_IO	-1765328214L	入出力エラー (リプレイのキャッシュ入出力コードで)
KRB5_RC_IO_UNKNOWN	-1765328213L	一般的な不明な RC/入出力エラー
KRB5_RC_IO_SPACE	-1765328212L	リプレイの情報を格納するためのシステム領域が不足しています
KRB5_TRANS_CANTOPEN	-1765328211L	レルム変換ファイルが開けません (または、見つかりません)
KRB5_TRANS_BADFORMAT	-1765328210L	レルム変換ファイルの形式が不適切です
KRB5_LNAME_CANTOPEN	-1765328209L	`lname` 変換データベースが開けません (または、見つかりません)
KRB5_LNAME_NOTRANS	-1765328208L	要求された主体で使用できる変換が存在しません
KRB5_LNAME_BADFORMAT	-1765328207L	変換データベースエントリの形式が不適切です
KRB5_CRYPTO_INTERNAL	-1765328206L	暗号システム内部エラー
KRB5_KT_BADNAME	-1765328205L	鍵テーブル名の形式が間違っています
KRB5_KT_UNKNOWN_TYPE	-1765328204L	鍵テーブル型が不明です

表 B–9 Kerberos v5 状態コード 5 *(続き)*

マイナー状態	値	意味
KRB5_KT_NOTFOUND	-1765328203L	Key table entry not found (鍵テーブルエントリが見つかりません。)
KRB5_KT_END	-1765328202L	鍵テーブルの終わりに到達しました
KRB5_KT_NOWRITE	-1765328201L	指定された鍵テーブルに書き込めません

Kerberos v5 で状態コード 6 として戻されるメッセージ

次の表に、Kerberos v5 で状態コード 6 として戻されるマイナー状態メッセージのリストを示します。

表 B–10 Kerberos v5 状態コード 6

マイナー状態	値	意味
KRB5_KT_IOERR	-1765328200L	鍵テーブルへの書き込み中にエラーが発生しました
KRB5_NO_TKT_IN_RLM	-1765328199L	要求されたレルムのチケットが見つかりません
KRB5DES_BAD_KEYPAR	-1765328198L	DES 鍵のパリティーが不良です
KRB5DES_WEAK_KEY	-1765328197L	DES 鍵が弱い鍵です
KRB5_BAD_ENCTYPE	-1765328196L	暗号化型が不良です
KRB5_BAD_KEYSIZE	-1765328195L	鍵サイズが暗号化型と互換性がありません
KRB5_BAD_MSIZE	-1765328194L	メッセージサイズが暗号化型と互換性がありません
KRB5_CC_TYPE_EXISTS	-1765328193L	資格キャッシュ型がすでに登録されています
KRB5_KT_TYPE_EXISTS	-1765328192L	鍵テーブル型がすでに登録されています

表 B–10 Kerberos v5 状態コード 6 *(続き)*

マイナー状態	値	意味
KRB5_CC_IO	-1765328191L	資格キャッシュ入出力操作が失敗しました
KRB5_FCC_PERM	-1765328190L	Credentials cache file permissions incorrect (資格キャッシュファイルのアクセス権が正しくありません。)
KRB5_FCC_NOFILE	-1765328189L	No credentials cache file found (資格キャッシュファイルが見つかりません。)
KRB5_FCC_INTERNAL	-1765328188L	内部ファイル資格キャッシュエラー
KRB5_CC_WRITE	-1765328187L	資格キャッシュファイルの書き込み中にエラーが発生しました
KRB5_CC_NOMEM	-1765328186L	資格キャッシュコードでこれ以上メモリーを割り当てられません
KRB5_CC_FORMAT	-1765328185L	資格キャッシュの形式が不良です
KRB5_INVALID_FLAGS	-1765328184L	KDC オプションの組み合わせが無効です (内部ライブラリエラー)
KRB5_NO_2ND_TKT	-1765328183L	要求に 2 番目のチケットが指定されていません
KRB5_NOCREDS_SUPPLIED	-1765328182L	ライブラリルーチンに資格が提供されていません
KRB5_SENDAUTH_BADAUTHVERS	-1765328181L	無効な sendauth バージョンが送信されました。
KRB5_SENDAUTH_BADAPPLVERS	-1765328180L	sendauth によって送信されたアプリケーションのバージョンが不良です
KRB5_SENDAUTH_BADRESPONSE	-1765328179L	sendauth の交換中の応答が不良です

表 B–10 Kerberos v5 状態コード 6 *(続き)*

マイナー状態	値	意味
KRB5_SENDAUTH_REJECTED	-1765328178L	sendauth の交換中にサーバーが認証を拒否しました

Kerberos v5 で状態コード 7 として戻されるメッセージ

次の表に、Kerberos v5 で状態コード 7 として戻されるマイナー状態メッセージのリストを示します。

表 B–11 Kerberos v5 状態コード 7

マイナー状態	値	意味
KRB5_PREAUTH_BAD_TYPE	-1765328177L	事前認証型がサポートされていません
KRB5_PREAUTH_NO_KEY	-1765328176L	要求された事前認証鍵が提供されていません
KRB5_PREAUTH_FAILED	-1765328175L	事前認証が失敗しました (一般的なエラー)
KRB5_RCACHE_BADVNO	-1765328174L	リプレイのキャッシュの形式のバージョン番号がサポートされていません
KRB5_CCACHE_BADVNO	-1765328173L	資格キャッシュの形式のバージョン番号がサポートされていません
KRB5_KEYTAB_BADVNO	-1765328172L	鍵テーブルの形式のバージョン番号がサポートされていません
KRB5_PROG_ATYPE_NOSUPP	-1765328171L	プログラムがアドレス型をサポートしていません
KRB5_RC_REQUIRED	-1765328170L	メッセージ再送検出には rcache パラメータが必要です。
KRB5_ERR_BAD_HOSTNAME	-1765328169L	ホスト名を標準化できません

表 B–11 Kerberos v5 状態コード 7 *(続き)*

マイナー状態	値	意味
KRB5_ERR_HOST_REALM_UNKNOWN	-1765328168L	Cannot determine realm for host (ホスト用のレルムを決定できません。)
KRB5_SNAME_UNSUPP_NAMETYPE	-1765328167L	名前型におけるサービス主体への変換が定義されていません
KRB5KRB_AP_ERR_V4_REPLY	-1765328166L	初期チケットの応答が Version 4 のエラーを示しています
KRB5_REALM_CANT_RESOLVE	-1765328165L	Cannot resolve KDC for requested realm (要求されたレルムの KDC を解決できません。)
KRB5_TKT_NOT_FORWARDABLE	-1765328164L	要求しているチケットは転送可能なチケットを取得できません
KRB5_FWD_BAD_PRINCIPAL	-1765328163L	(資格の転送中) 主体名が不良です
KRB5_GET_IN_TKT_LOOP	-1765328162L	Looping detected inside krb5_get_in_tkt (krb5_get_in_tkt 内部でループが検出されました。)
KRB5_CONFIG_NODEFREALM	-1765328161L	Kerberos 構成ファイル `/etc/krb5/krb5.conf` がデフォルトのレルムを指定していません
KRB5_SAM_UNSUPPORTED	-1765328160L	obtain_sam_padata に無効な SAM フラグがあります。
KRB5_KT_NAME_TOOLONG	-1765328159L	鍵タブ名が長すぎます
KRB5_KT_KVNONOTFOUND	-1765328158L	Key version number for principal in key table is incorrect (鍵テーブルの主体の鍵バージョン番号が正しくありません。)
KRB5_CONF_NOT_CONFIGURED	-1765328157L	Kerberos 構成ファイル `/etc/krb5/krb5.conf` が構成されていません

表 B–11 Kerberos v5 状態コード 7 *(続き)*

マイナー状態	値	意味
ERROR_TABLE_BASE_krb5	-1765328384L	default

◆ ◆ ◆ 付録 C

OIDの指定

できるだけGSS-APIが提供するデフォルトのQOPおよび機構を使用してください。70ページの「GSS-APIのOID」を参照してください。ただし、何らかの理由でQOPのOIDを指定する必要がある場合があります。この付録では、OIDを指定する方法について説明します。この章の内容は次のとおりです。

OID値が含まれるファイル

GSS-APIでは、機構とQOPを人が読める形式で表示することができます。Solarisシステムでは、`/etc/gss/mech`と`/etc/gss/qop`の2つのファイルに、使用できる機構とQOPについての情報が含まれています。この2つのファイルへのアクセス権がない場合は、他のソースから文字列リテラルを提供する必要があります。その機構やQOP用に公開されているインターネット標準は、この目的にかなっています。

/etc/gss/mechファイル

`/etc/gss/mech`ファイルには、使用できる機構のリストが含まれています。`/etc/gss/mech`には、機構名が数値とアルファベットの両方の形式で格納されています。`/etc/gss/mech`の各行は、次の書式で構成されています。

- 機構名(ASCII文字列)
- 機構のOID
- その機構によって提供されるサービスを実装するための共有ライブラリ
- サービスを実装するためのカーネルモジュール(オプション)

`Example C–1`に、例C–1ファイルの例を示します。

例 C–1 /etc/gss/mech ファイル

```
#
# Copyright 2003 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
#ident    "@(#)mech    1.12    03/10/20 SMI"
#
# This file contains the GSS-API based security mechanism names,
# the associated object identifiers (OID) and a shared library that
# implements the services for the mechanisms under GSS-API.
#
# Mechanism Name    Object Identifier    Shared Library    Kernel Module
[Options]
#
kerberos_v5        1.2.840.113554.1.2.2    mech_krb5.so kmech_krb5
spnego            1.3.6.1.5.5.2        mech_spnego.so.1 [msinterop]
diffie_hellman_640_0    1.3.6.4.1.42.2.26.2.4    dh640-0.so.1
diffie_hellman_1024_0    1.3.6.4.1.42.2.26.2.5    dh1024-0.so.1
```

/etc/gss/qop ファイル

/etc/gss/qop ファイルには、導入されたすべての機構用に、各機構がサポートするすべての QOP が、ASCII 文字列とそれに対応する 32 ビット整数の両方で格納されます。次に、/etc/gss/qop ファイルの例を示します。

例 C–2 /etc/gss/qop ファイル

```
#
# Copyright (c) 2000, by Sun Microsystems, Inc.
# All rights reserved.
#
#ident  "@(#)qop 1.3     00/11/09 SMI"
#
# This file contains information about the GSS-API based quality of
# protection (QOP), its string name and its value (32-bit integer).
#
# QOP string                    QOP Value       Mechanism Name
#
GSS_KRB5_INTEG_C_QOP_DES_MD5    0               kerberos_v5
GSS_KRB5_CONF_C_QOP_DES         0               kerberos_v5
```

gss_str_to_oid() 関数

旧バージョンの GSS-API との下位互換性のため、この実装の GSS-API は gss_str_to_oid() 関数をサポートします。gss_str_to_oid() は、機構または QOP を表す文字列を OID に変換します。この文字列は、数値または ASCII 文字列のどちらでもかまいません。

注意 – デフォルトの機構と QOP を使用することが強く推奨されているため、`gss_str_to_oid()`、`gss_oid_to_str()`、および `gss_release_oid()` をサポートしていない GSS-API の実装もあります。

機構を表す文字列は、アプリケーション内でハードコード化することも、ユーザー入力から取得することも可能です。しかし、必ずしもすべての GSS-API の実装が `gss_str_to_oid()` 関数をサポートしているわけではないため、アプリケーションはこの関数に依存すべきではありません。

機構を表す数値には、2 つの異なる形式を指定できます。1 つは `{ 1 2 3 4 }` であり、GSS-API 仕様によって正式に認められています。もう 1 つは `1.2.3.4` で、こちらの方が広く使用されていますが、正式な標準形式ではありません。`gss_str_to_oid()` は機構の数値として最初の形式を期待します。したがって、2 番目の形式を使用している場合は、`gss_str_to_oid()` を呼び出す前に 1 番目の形式に変換する必要があります。`gss_str_to_oid()` の例については、例 C–3 を参照してください。機構が有効でない場合、`gss_str_to_oid()` は `GSS_S_BAD_MECH` を戻します。

`gss_str_to_oid()` は GSS-API データ領域を割り当てるため、終了時には、割り当てられた OID を `gss_release_oid()` 関数で削除する必要があります。`gss_str_to_oid()` と同様に、`gss_release_oid()` も一般的にサポートされている関数ではありません。したがって、移植性を最大限にしたいプログラムはこの関数に依存すべきではありません。

機構 OID の構築

`gss_str_to_oid()` は常に使用できるわけではないため、機構を調べて選択する方法がほかにもいくつかあります。1 つは、機構 OID を手動で構築し、その機構を使用できる機構の集合と比較する方法です。もう 1 つは、使用できる機構の集合を取得して、その中から 1 つを選択する方法です。

次に、`gss_OID` 型の形式を示します。

```
typedef struct gss_OID_desc struct {
     OM_uint32 length;
     void           *elements;
} gss_OID_desc, *gss_OID;
```

この構造体の *elements* フィールドは、`gss_OID` の通常の BER TLV エンコーディングの値の部分の ASN.1 BER エンコーディングが格納されているオクテット文字列の最初のバイトを指します。*length* フィールドには、この値のバイト数が格納されています。たとえば、DASS X.509 認証機構に対応する `gss_OID` 値の場合、*length* フィールドは 7 で、*elements* フィールドは 8 進数で「`53,14,2,207,163,7,5`」となる値を指します。

機構 OID を構築する 1 つの方法は、gss_OID を宣言し、次にその要素を手動で初期化して、その機構を表すようにします。前述のとおり、*elements* 値はハードコード化することも、表から取得することも、ユーザーが入力することもできます。この方法は、gss_str_to_oid() を使用するよりも手がかかりますが、同じ効果が得られます。

次に、手動で構築した gss_OID を、使用できる機構の集合と比較します。使用できる機構の集合は、gss_indicate_mechs() または gss_inquire_mechs_for_name() の関数から戻されたものです。手動で構築した機構 OID が、使用できる機構の集合の中に存在するかどうかを調べるには、gss_test_oid_set_member() 関数を使用します。gss_test_oid_set_member() がエラーを戻さなかった場合、手動で構築した OID は GSS-API トランザクション用の機構として使用できます。

OID を手動で構築する代わりに、gss_indicate_mechs() または gss_inquire_mechs_for_name() を使用すると、使用できる機構の gss_OID_set を取得できます。次に、gss_OID_set の形式を示します。

```
typedef struct gss_OID_set_desc_struct {
     OM_uint32 length;
     void           *elements;
} gss_OID_set_desc, *gss_OID_set;
```

elements は機構を表す gss_OID です。アプリケーションは、各機構を解析し、それぞれの数値表現を表示できます。ユーザーはこの表示を使用して機構を選択できます。次に、アプリケーションは選択した機構が gss_OID_set の適切なメンバーになるように設定します。また、希望する機構と使用できる機構のリストを比較することもできます。

createMechOid() 関数

この関数は、完全を期すためにここに表示されています。通常は、GSS_C_NULL_OID によって指定されるデフォルトの機構を使用するようにしてください。

例 C–3 createMechOid() 関数

```
gss_OID createMechOid(const char *mechStr)
{
        gss_buffer_desc mechDesc;
        gss_OID mechOid;
        OM_uint32 minor;

        if (mechStr == NULL)
                return (GSS_C_NULL_OID);

        mechDesc.length = strlen(mechStr);
        mechDesc.value = (void *) mechStr;

        if (gss_str_to_oid(&minor, &mechDesc, &mechOid) !
            = GSS_S_COMPLETE) {
```

例 C–3 createMechOid() 関数 (続き)

```
                fprintf(stderr, "Invalid mechanism oid specified <%s>",
                                mechStr);
                return (GSS_C_NULL_OID);
        }

        return (mechOid);
}
```

デフォルト以外の機構の指定

parse_oid() は、コマンド行で指定されたセキュリティー機構名を互換性のある OID に変換します。

例 C–4 parse_oid() 関数

```
static void parse_oid(char *mechanism, gss_OID *oid)
{
    char        *mechstr = 0, *cp;
    gss_buffer_desc tok;
    OM_uint32 maj_stat, min_stat;

    if (isdigit(mechanism[0])) {
        mechstr = malloc(strlen(mechanism)+5);
        if (!mechstr) {
            printf("Couldn't allocate mechanism scratch!\n");
            return;
        }
        sprintf(mechstr, "{ %s }", mechanism);
        for (cp = mechstr; *cp; cp++)
            if (*cp == '.')
                *cp = ' ';
        tok.value = mechstr;
    } else
        tok.value = mechanism;
    tok.length = strlen(tok.value);
    maj_stat = gss_str_to_oid(&min_stat, &tok, oid);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("str_to_oid", maj_stat, min_stat);
        return;
    }
    if (mechstr)
        free(mechstr);
}
```

◆ ◆ ◆ 付 録 D

SASL ソースコード例

この付録には、146 ページの「SASL の例」に記載されたソースコード例が収められています。この付録は、次のような節から構成されています。

- 261 ページの「SASL クライアントの例」
- 269 ページの「SASL サーバーの例」
- 277 ページの「共通のコード」

SASL クライアントの例

次のコードリストは、146 ページの「SASL の例」に記載されているクライアントの例を対象としています。

このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

```
#pragma ident    "@(#)client.c    1.4    03/04/07 SMI"
/* $Id: client.c,v 1.3 2002/09/03 15:11:59 rjs3 Exp $ */
/*
 * Copyright (c) 2001 Carnegie Mellon University.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The name "Carnegie Mellon University" must not be used to
 *    endorse or promote products derived from this software without
```

```
 *    prior written permission. For permission or any other legal
 *    details, please contact
 *      Office of Technology Transfer
 *      Carnegie Mellon University
 *      5000 Forbes Avenue
 *      Pittsburgh, PA  15213-3890
 *      (412) 268-4387, fax: (412) 268-7395
 *      tech-transfer@andrew.cmu.edu
 *
 * 4. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by Computing Services
 *     at Carnegie Mellon University (http://www.cmu.edu/computing/)."
 *
 * CARNEGIE MELLON UNIVERSITY DISCLAIMS ALL WARRANTIES WITH REGARD TO
 * THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS, IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY BE LIABLE
 * FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN
 * AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
 * OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */

#include <config.h>

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <ctype.h>
#include <errno.h>
#include <string.h>

#ifdef HAVE_UNISTD_H
#include <unistd.h>
#endif

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#ifdef _SUN_SDK_
#include <sysexits.h>
#endif /* _SUN_SDK_ */

#include <assert.h>

#include <sasl.h>

#include "common.h"

/* remove \r\n at end of the line */
static void chop(char *s)
{
    char *p;

    assert(s);
    p = s + strlen(s) - 1;
    if (p[0] == '\n') {
    *p-- = '\0';
```

```
    }
    if (p >= s && p[0] == '\r') {
    *p-- = '\0';
    }
}

static int getrealm(void *context __attribute__((unused)),
            int id,
            const char **availrealms,
            const char **result)
{
    static char buf[1024];

    /* Double-check the ID */
    if (id != SASL_CB_GETREALM) return SASL_BADPARAM;
    if (!result) return SASL_BADPARAM;

    printf("please choose a realm (available:");
    while (*availrealms) {
    printf(" %s", *availrealms);
    availrealms++;
    }
    printf("): ");

    fgets(buf, sizeof buf, stdin);
    chop(buf);
    *result = buf;

    return SASL_OK;
}

static int simple(void *context __attribute__((unused)),
          int id,
          const char **result,
          unsigned *len)
{
    static char buf[1024];

    /* Double-check the connection */
    if (! result)
    return SASL_BADPARAM;

    switch (id) {
    case SASL_CB_USER:
    printf("please enter an authorization id: ");
    break;
    case SASL_CB_AUTHNAME:
    printf("please enter an authentication id: ");
    break;
    default:
    return SASL_BADPARAM;
    }

    fgets(buf, sizeof buf, stdin);
    chop(buf);
    *result = buf;
    if (len) *len = strlen(buf);

    return SASL_OK;
```

```
}

#ifndef HAVE_GETPASSPHRASE
static char *
getpassphrase(const char *prompt)
{
  return getpass(prompt);
}
#endif /* ! HAVE_GETPASSPHRASE */

static int
getsecret(sasl_conn_t *conn,
      void *context __attribute__((unused)),
      int id,
      sasl_secret_t **psecret)
{
    char *password;
    size_t len;
    static sasl_secret_t *x;

    /* paranoia check */
    if (! conn || ! psecret || id != SASL_CB_PASS)
    return SASL_BADPARAM;

    password = getpassphrase("Password: ");
    if (! password)
    return SASL_FAIL;

    len = strlen(password);

    x = (sasl_secret_t *) realloc(x, sizeof(sasl_secret_t) + len);

    if (!x) {
    memset(password, 0, len);
    return SASL_NOMEM;
    }

    x->len = len;
#ifdef _SUN_SDK_
    strcpy((char *)x->data, password);
#else
    strcpy(x->data, password);
#endif /* _SUN_SDK_ */
    memset(password, 0, len);

    *psecret = x;
    return SASL_OK;
}

static int getpath(void * context __attribute__((unused)),
    const char **path)
{
    *path = getenv("SASL_PATH");

    if (*path == NULL)
    *path = PLUGINDIR;

    return SASL_OK;
}
```

```
/* callbacks we support */
static sasl_callback_t callbacks[] = {
  {
    SASL_CB_GETREALM, &getrealm, NULL
  }, {
    SASL_CB_USER, &simple, NULL
  }, {
    SASL_CB_AUTHNAME, &simple, NULL
  }, {
    SASL_CB_PASS, &getsecret, NULL
  }, {
    SASL_CB_GETPATH, &getpath, NULL
  }, {
    SASL_CB_LIST_END, NULL, NULL
  }
};

int getconn(const char *host, const char *port)
{
    struct addrinfo hints, *ai, *r;
    int err, sock = -1;

    memset(&hints, 0, sizeof(hints));
    hints.ai_family = PF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;

    if ((err = getaddrinfo(host, port, &hints, &ai)) != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(err));
    exit(EX_UNAVAILABLE);
    }

    for (r = ai; r; r = r->ai_next) {
    sock = socket(r->ai_family, r->ai_socktype, r->ai_protocol);
    if (sock < 0)
        continue;
    if (connect(sock, r->ai_addr, r->ai_addrlen) >= 0)
        break;
    close(sock);
    sock = -1;
    }

    freeaddrinfo(ai);
    if (sock < 0) {
    perror("connect");
    exit(EX_UNAVAILABLE);
    }

    return sock;
}

char *mech;

int mysasl_negotiate(FILE *in, FILE *out, sasl_conn_t *conn)
{
    char buf[8192];
    const char *data;
    const char *chosenmech;
#ifdef _SUN_SDK_
```

```
    unsigned len;
#else
    int len;
#endif /* _SUN_SDK_ */
    int r, c;

    /* get the capability list */
    dprintf(0, "receiving capability list... ");
    len = recv_string(in, buf, sizeof buf);
    dprintf(0, "%s\n", buf);

    if (mech) {
    /* make sure that 'mech' appears in 'buf' */
    if (!strstr(buf, mech)) {
        printf("server doesn't offer mandatory mech '%s'\n", mech);
        return -1;
    }
    } else {
    mech = buf;
    }

    r = sasl_client_start(conn, mech, NULL, &data, &len, &chosenmech);
    if (r != SASL_OK && r != SASL_CONTINUE) {
    saslerr(r, "starting SASL negotiation");
    printf("\n%s\n", sasl_errdetail(conn));
    return -1;
    }

    dprintf(1, "using mechanism %s\n", chosenmech);

    /* we send up to 3 strings;
       the mechanism chosen, the presence of initial response,
       and optionally the initial response */
    send_string(out, chosenmech, strlen(chosenmech));
    if(data) {
    send_string(out, "Y", 1);
    send_string(out, data, len);
    } else {
    send_string(out, "N", 1);
    }

    for (;;) {
    dprintf(2, "waiting for server reply...\n");

    c = fgetc(in);
    switch (c) {
    case 'O':
        goto done_ok;

    case 'N':
        goto done_no;

    case 'C': /* continue authentication */
        break;

    default:
        printf("bad protocol from server (%c %x)\n", c, c);
        return -1;
    }
```

```
    len = recv_string(in, buf, sizeof buf);

    r = sasl_client_step(conn, buf, len, NULL, &data, &len);
    if (r != SASL_OK && r != SASL_CONTINUE) {
        saslerr(r, "performing SASL negotiation");
        printf("\n%s\n", sasl_errdetail(conn));
        return -1;
    }

    if (data) {
        dprintf(2, "sending response length %d...\n", len);
        send_string(out, data, len);
    } else {
        dprintf(2, "sending null response...\n");
        send_string(out, "", 0);
    }
    }

 done_ok:
    printf("successful authentication\n");
    return 0;

 done_no:
    printf("authentication failed\n");
    return -1;
}

#ifdef _SUN_SDK_
void usage(const char *s)
#else
void usage(void)
#endif /* _SUN_SDK_ */
{
#ifdef _SUN_SDK_
    fprintf(stderr, "usage: %s [-p port] [-s service] [-m mech] host\n", s);
#else
    fprintf(stderr, "usage: client [-p port] [-s service] \
        [-m mech] host\n");
#endif /* _SUN_SDK_ */
    exit(EX_USAGE);
}

int main(int argc, char *argv[])
{
    int c;
    char *host = "localhost";
    char *port = "12345";
    char localaddr[NI_MAXHOST + NI_MAXSERV],
    remoteaddr[NI_MAXHOST + NI_MAXSERV];
    char *service = "rcmd";
    char hbuf[NI_MAXHOST], pbuf[NI_MAXSERV];
    int r;
    sasl_conn_t *conn;
    FILE *in, *out;
    int fd;
    int salen;
    struct sockaddr_storage local_ip, remote_ip;

    while ((c = getopt(argc, argv, "p:s:m:")) != EOF) {
```

```
    switch(c) {
    case 'p':
        port = optarg;
        break;

    case 's':
        service = optarg;
        break;

    case 'm':
        mech = optarg;
        break;

    default:
#ifdef _SUN_SDK_
        usage(argv[0]);
#else
        usage();
#endif /* _SUN_SDK_ */
        break;
    }
    }

    if (optind > argc - 1) {
#ifdef _SUN_SDK_
    usage(argv[0]);
#else
    usage();
#endif /* _SUN_SDK_ */
    }
    if (optind == argc - 1) {
    host = argv[optind];
    }

    /* initialize the sasl library */
    r = sasl_client_init(callbacks);
    if (r != SASL_OK) saslfail(r, "initializing libsasl");

    /* connect to remote server */
    fd = getconn(host, port);

    /* set ip addresses */
    salen = sizeof(local_ip);
    if (getsockname(fd, (struct sockaddr *)&local_ip, &salen) < 0) {
    perror("getsockname");
    }

    getnameinfo((struct sockaddr *)&local_ip, salen,
        hbuf, sizeof(hbuf), pbuf, sizeof(pbuf),
#ifdef _SUN_SDK_    /* SOLARIS doesn't support NI_WITHSCOPEID */
        NI_NUMERICHOST | NI_NUMERICSERV);
#else
        NI_NUMERICHOST | NI_WITHSCOPEID | NI_NUMERICSERV);
#endif
    snprintf(localaddr, sizeof(localaddr), "%s;%s", hbuf, pbuf);

    salen = sizeof(remote_ip);
    if (getpeername(fd, (struct sockaddr *)&remote_ip, &salen) < 0) {
    perror("getpeername");
```

```
    }

    getnameinfo((struct sockaddr *)&remote_ip, salen,
        hbuf, sizeof(hbuf), pbuf, sizeof(pbuf),
#ifdef _SUN_SDK_    /* SOLARIS doesn't support NI_WITHSCOPEID */
        NI_NUMERICHOST | NI_NUMERICSERV);
#else
        NI_NUMERICHOST | NI_WITHSCOPEID | NI_NUMERICSERV);
#endif
    snprintf(remoteaddr, sizeof(remoteaddr), "%s;%s", hbuf, pbuf);

    /* client new connection */
    r = sasl_client_new(service, host, localaddr, remoteaddr, NULL,
        0, &conn);
    if (r != SASL_OK) saslfail(r, "allocating connection state");

    /* set external properties here
       sasl_setprop(conn, SASL_SSF_EXTERNAL, &extprops); */

    /* set required security properties here
       sasl_setprop(conn, SASL_SEC_PROPS, &secprops); */

    in = fdopen(fd, "r");
    out = fdopen(fd, "w");

    r = mysasl_negotiate(in, out, conn);
    if (r == SASL_OK) {
    /* send/receive data */


    }

    printf("closing connection\n");
    fclose(in);
    fclose(out);
    close(fd);
    sasl_dispose(&conn);

    sasl_done();

    return 0;
}
```

SASL サーバーの例

次のコードリストは、146 ページの「SASL の例」に記載されているサーバーの例を対象としています。

このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

```
#pragma ident    "@(#)server.c    1.3    03/04/07 SMI"
/* $Id: server.c,v 1.4 2002/10/07 05:04:05 rjs3 Exp $ */
/*
```

```
 * Copyright (c) 2001 Carnegie Mellon University.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The name "Carnegie Mellon University" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For permission or any other legal
 *    details, please contact
 *      Office of Technology Transfer
 *      Carnegie Mellon University
 *      5000 Forbes Avenue
 *      Pittsburgh, PA  15213-3890
 *      (412) 268-4387, fax: (412) 268-7395
 *      tech-transfer@andrew.cmu.edu
 *
 * 4. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by Computing Services
 *     at Carnegie Mellon University (http://www.cmu.edu/computing/)."
 *
 * CARNEGIE MELLON UNIVERSITY DISCLAIMS ALL WARRANTIES WITH REGARD TO
 * THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS, IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY BE LIABLE
 * FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN
 * AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
 * OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */

#include <config.h>

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <ctype.h>
#include <errno.h>
#include <string.h>

#ifdef HAVE_UNISTD_H
#include <unistd.h>
#endif

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#ifdef _SUN_SDK_
#include <sysexits.h>
#endif /* _SUN_SDK_ */
```

```
#include <sasl.h>

#include "common.h"

#if !defined(IPV6_BINDV6ONLY) && defined(IN6P_IPV6_V6ONLY)
#define IPV6_BINDV6ONLY IN6P_BINDV6ONLY
#endif
#if !defined(IPV6_V6ONLY) && defined(IPV6_BINDV6ONLY)
#define    IPV6_V6ONLY    IPV6_BINDV6ONLY
#endif
#ifndef IPV6_BINDV6ONLY
#undef      IPV6_V6ONLY
#endif

static int getpath(void * context __attribute__((unused)),
    const char **path)
{
    *path = getenv("SASL_PATH");

    if (*path == NULL)
    *path = PLUGINDIR;

    return SASL_OK;
}

/* callbacks we support */
static sasl_callback_t callbacks[] = {
  {
    SASL_CB_GETPATH, &getpath, NULL
  }, {
    SASL_CB_LIST_END, NULL, NULL
  }
};

/* create a socket listening on port 'port' */
/* if af is PF_UNSPEC more than one socket might be returned */
/* the returned list is dynamically allocated, so caller needs to free it */
int *listensock(const char *port, const int af)
{
    struct addrinfo hints, *ai, *r;
    int err, maxs, *sock, *socks;
    const int on = 1;

    memset(&hints, 0, sizeof(hints));
    hints.ai_flags = AI_PASSIVE;
    hints.ai_family = af;
    hints.ai_socktype = SOCK_STREAM;
    err = getaddrinfo(NULL, port, &hints, &ai);
    if (err) {
    fprintf(stderr, "%s\n", gai_strerror(err));
    exit(EX_USAGE);
    }

    /* Count max number of sockets we can open */
    for (maxs = 0, r = ai; r; r = r->ai_next, maxs++)
    ;
    socks = malloc((maxs + 1) * sizeof(int));
    if (!socks) {
```

```
    fprintf(stderr, "couldn't allocate memory for sockets\n");
    freeaddrinfo(ai);
    exit(EX_OSERR);
    }

    socks[0] = 0;    /* num of sockets counter at start of array */
    sock = socks + 1;
    for (r = ai; r; r = r->ai_next) {
    fprintf(stderr, "trying %d, %d, %d\n",r->ai_family, r->ai_socktype,
     r->ai_protocol);
    *sock = socket(r->ai_family, r->ai_socktype, r->ai_protocol);
    if (*sock < 0) {
        perror("socket");
        continue;
    }
    if (setsockopt(*sock, SOL_SOCKET, SO_REUSEADDR,
               (void *) &on, sizeof(on)) < 0) {
        perror("setsockopt(SO_REUSEADDR)");
        close(*sock);
        continue;
    }
#if defined(IPV6_V6ONLY) && !(defined(__FreeBSD__) && __FreeBSD__ < 3)
    if (r->ai_family == AF_INET6) {
        if (setsockopt(*sock, IPPROTO_IPV6, IPV6_BINDV6ONLY,
               (void *) &on, sizeof(on)) < 0) {
        perror("setsockopt (IPV6_BINDV6ONLY)");
        close(*sock);
        continue;
        }
    }
#endif
    if (bind(*sock, r->ai_addr, r->ai_addrlen) < 0) {
        perror("bind");
        close(*sock);
        continue;
     }

     if (listen(*sock, 5) < 0) {
         perror("listen");
         close(*sock);
         continue;
     }

     socks[0]++;
     sock++;
    }

    freeaddrinfo(ai);

    if (socks[0] == 0) {
     fprintf(stderr, "Couldn't bind to any socket\n");
     free(socks);
    exit(EX_OSERR);
    }

    return socks;
}

#ifdef _SUN_SDK_
```

```
void usage(const char *s)
#else
void usage(void)
#endif /* _SUN_SDK_ */
{
#ifdef _SUN_SDK_
    fprintf(stderr, "usage: %s [-p port] [-s service] [-m mech]\n", s);
#else
    fprintf(stderr, "usage: server [-p port] [-s service] [-m mech]\n");
#endif /* _SUN_SDK_ */
    exit(EX_USAGE);
}

/* Globals are used here, but local variables are preferred */
char *mech;

/* do the sasl negotiation; return -1 if it fails */
int mysasl_negotiate(FILE *in, FILE *out, sasl_conn_t *conn)
{
    char buf[8192];
    char chosenmech[128];
    const char *data;
#ifdef _SUN_SDK_
    unsigned len;
#else
    int len;
#endif /* _SUN_SDK_ */
    int r = SASL_FAIL;
    const char *userid;

    /* generate the capability list */
    if (mech) {
    dprintf(2, "forcing use of mechanism %s\n", mech);
    data = strdup(mech);
    } else {
    int count;

    dprintf(1, "generating client mechanism list... ");
    r = sasl_listmech(conn, NULL, NULL, " ", NULL,
              &data, &len, &count);
    if (r != SASL_OK) saslfail(r, "generating mechanism list");
    dprintf(1, "%d mechanisms\n", count);
    }

    /* send capability list to client */
    send_string(out, data, len);

    dprintf(1, "waiting for client mechanism...\n");
    len = recv_string(in, chosenmech, sizeof chosenmech);
    if (len <= 0) {
    printf("client didn't choose mechanism\n");
    fputc('N', out); /* send NO to client */
    fflush(out);
    return -1;
    }

    if (mech && strcasecmp(mech, chosenmech)) {
    printf("client didn't choose mandatory mechanism\n");
    fputc('N', out); /* send NO to client */
```

```
fflush(out);
return -1;
}

len = recv_string(in, buf, sizeof(buf));
if(len != 1) {
saslerr(r, "didn't receive first-send parameter correctly");
fputc('N', out);
fflush(out);
return -1;
}

if(buf[0] == 'Y') {
    /* receive initial response (if any) */
    len = recv_string(in, buf, sizeof(buf));

    /* start libsasl negotiation */
    r = sasl_server_start(conn, chosenmech, buf, len,
              &data, &len);
} else {
r = sasl_server_start(conn, chosenmech, NULL, 0,
              &data, &len);
}

if (r != SASL_OK && r != SASL_CONTINUE) {
saslerr(r, "starting SASL negotiation");
fputc('N', out); /* send NO to client */
fflush(out);
return -1;
}

while (r == SASL_CONTINUE) {
if (data) {
    dprintf(2, "sending response length %d...\n", len);
    fputc('C', out); /* send CONTINUE to client */
    send_string(out, data, len);
} else {
    dprintf(2, "sending null response...\n");
    fputc('C', out); /* send CONTINUE to client */
    send_string(out, "", 0);
}

dprintf(1, "waiting for client reply...\n");
len = recv_string(in, buf, sizeof buf);
if (len < 0) {
    printf("client disconnected\n");
    return -1;
}

r = sasl_server_step(conn, buf, len, &data, &len);
if (r != SASL_OK && r != SASL_CONTINUE) {
    saslerr(r, "performing SASL negotiation");
    fputc('N', out); /* send NO to client */
    fflush(out);
    return -1;
}
}

if (r != SASL_OK) {
```

```
    saslerr(r, "incorrect authentication");
    fputc('N', out); /* send NO to client */
    fflush(out);
    return -1;
    }

    fputc('O', out); /* send OK to client */
    fflush(out);
    dprintf(1, "negotiation complete\n");

    r = sasl_getprop(conn, SASL_USERNAME, (const void **) &userid);
    printf("successful authentication '%s'\n", userid);

    return 0;
}

int main(int argc, char *argv[])
{
    int c;
    char *port = "12345";
    char *service = "rcmd";
    int *l, maxfd=0;
    int r, i;
    sasl_conn_t *conn;

    while ((c = getopt(argc, argv, "p:s:m:")) != EOF) {
    switch(c) {
    case 'p':
        port = optarg;
        break;

    case 's':
        service = optarg;
        break;

    case 'm':
        mech = optarg;
        break;

    default:
#ifdef _SUN_SDK_
        usage(argv[0]);
#else
        usage();
#endif /* _SUN_SDK_ */
        break;
    }
    }

    /* initialize the sasl library */
    r = sasl_server_init(callbacks, "sample");
    if (r != SASL_OK) saslfail(r, "initializing libsasl");

    /* get a listening socket */
    if ((l = listensock(port, PF_UNSPEC)) == NULL) {
    saslfail(SASL_FAIL, "allocating listensock");
    }

    for (i = 1; i <= l[0]; i++) {
```

```
        if (l[i] > maxfd)
            maxfd = l[i];
    }

    for (;;) {
    char localaddr[NI_MAXHOST | NI_MAXSERV],
         remoteaddr[NI_MAXHOST | NI_MAXSERV];
    char myhostname[1024+1];
    char hbuf[NI_MAXHOST], pbuf[NI_MAXSERV];
    struct sockaddr_storage local_ip, remote_ip;
    int salen;
    int nfds, fd = -1;
    FILE *in, *out;
    fd_set readfds;

    FD_ZERO(&readfds);
    for (i = 1; i <= l[0]; i++)
        FD_SET(l[i], &readfds);

    nfds = select(maxfd + 1, &readfds, 0, 0, 0);
    if (nfds <= 0) {
        if (nfds < 0 && errno != EINTR)
        perror("select");
        continue;
    }

       for (i = 1; i <= l[0]; i++)
           if (FD_ISSET(l[i], &readfds)) {
               fd = accept(l[i], NULL, NULL);
               break;
           }

    if (fd < 0) {
        if (errno != EINTR)
        perror("accept");
        continue;
    }

    printf("accepted new connection\n");

    /* set ip addresses */
    salen = sizeof(local_ip);
    if (getsockname(fd, (struct sockaddr *)&local_ip, &salen) < 0) {
        perror("getsockname");
    }
    getnameinfo((struct sockaddr *)&local_ip, salen,
            hbuf, sizeof(hbuf), pbuf, sizeof(pbuf),
#ifdef _SUN_SDK_    /* SOLARIS doesn't support NI_WITHSCOPEID */
            NI_NUMERICHOST | NI_NUMERICSERV);
#else
            NI_NUMERICHOST | NI_WITHSCOPEID | NI_NUMERICSERV);
#endif
    snprintf(localaddr, sizeof(localaddr), "%s;%s", hbuf, pbuf);

    salen = sizeof(remote_ip);
    if (getpeername(fd, (struct sockaddr *)&remote_ip, &salen) < 0) {
        perror("getpeername");
    }
```

```
    getnameinfo((struct sockaddr *)&remote_ip, salen,
            hbuf, sizeof(hbuf), pbuf, sizeof(pbuf),
#ifdef _SUN_SDK_    /* SOLARIS doesn't support NI_WITHSCOPEID */
            NI_NUMERICHOST | NI_NUMERICSERV);
#else
            NI_NUMERICHOST | NI_WITHSCOPEID | NI_NUMERICSERV);
#endif
    snprintf(remoteaddr, sizeof(remoteaddr), "%s;%s", hbuf, pbuf);

    r = gethostname(myhostname, sizeof(myhostname)-1);
    if(r == -1) saslfail(r, "getting hostname");

    r = sasl_server_new(service, myhostname, NULL, localaddr, remoteaddr,
                NULL, 0, &conn);
    if (r != SASL_OK) saslfail(r, "allocating connection state");

    /* set external properties here
       sasl_setprop(conn, SASL_SSF_EXTERNAL, &extprops); */

    /* set required security properties here
       sasl_setprop(conn, SASL_SEC_PROPS, &secprops); */

    in = fdopen(fd, "r");
    out = fdopen(fd, "w");

    r = mysasl_negotiate(in, out, conn);
    if (r == SASL_OK) {
        /* send/receive data */


    }

    printf("closing connection\n");
    fclose(in);
    fclose(out);
    close(fd);
    sasl_dispose(&conn);
    }

    sasl_done();
}
```

共通のコード

次のコード例には、その他の SASL 関数のリストが含まれています。

このソースコード例は、Sun ダウンロードセンターからダウンロードすることも可能です。http://www.sun.com/download/products.xml?id=41912db5 を参照してください。

```
#pragma ident    "@(#)common.c    1.1    03/03/28 SMI"
/* $Id: common.c,v 1.3 2002/09/03 15:11:59 rjs3 Exp $ */
/*
 * Copyright (c) 2001 Carnegie Mellon University.  All rights reserved.
```

```
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The name "Carnegie Mellon University" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For permission or any other legal
 *    details, please contact
 *      Office of Technology Transfer
 *      Carnegie Mellon University
 *      5000 Forbes Avenue
 *      Pittsburgh, PA  15213-3890
 *      (412) 268-4387, fax: (412) 268-7395
 *      tech-transfer@andrew.cmu.edu
 *
 * 4. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by Computing Services
 *     at Carnegie Mellon University (http://www.cmu.edu/computing/)."
 *
 * CARNEGIE MELLON UNIVERSITY DISCLAIMS ALL WARRANTIES WITH REGARD TO
 * THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS, IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY BE LIABLE
 * FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN
 * AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
 * OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */

#include <config.h>

#include <stdio.h>
#include <ctype.h>
#include <stdarg.h>
#ifdef _SUN_SDK_
#include <sysexits.h>
#endif /* _SUN_SDK_ */

#include <sasl.h>

/* send/recv library for IMAP4 style literals.

   really not important; just one way of doing length coded strings */

int send_string(FILE *f, const char *s, int l)
{
    int al;

    al = fprintf(f, "{%d}\r\n", l);
    fwrite(s, 1, l, f);
```

```
    fflush(f);

    printf("send: {%d}\n", l);
    while (l--) {
    if (isprint((unsigned char) *s)) {
        printf("%c", *s);
    } else {
        printf("[%X]", (unsigned char) *s);
    }
    s++;
    }
    printf("\n");

    return al;
}

int recv_string(FILE *f, char *buf, int buflen)
{
    int c;
    int len, l;
    char *s;

    c = fgetc(f);
    if (c != '{') return -1;

    /* read length */
    len = 0;
    c = fgetc(f);
    while (isdigit(c)) {
    len = len * 10 + (c - '0');
    c = fgetc(f);
    }
    if (c != '}') return -1;
    c = fgetc(f);
    if (c != '\r') return -1;
    c = fgetc(f);
    if (c != '\n') return -1;

    /* read string */
    if (buflen <= len) {
    fread(buf, buflen - 1, 1, f);
    buf[buflen - 1] = '\0';
    /* discard oversized string */
    len -= buflen - 1;
    while (len--) (void)fgetc(f);

    len = buflen - 1;
    } else {
    fread(buf, len, 1, f);
    buf[len] = '\0';
    }

    l = len;
    s = buf;
    printf("recv: {%d}\n", len);
    while (l--) {
    if (isprint((unsigned char) *s)) {
        printf("%c", *s);
    } else {
```

```
            printf("[%X]", (unsigned char) *s);
        }
        s++;
        }
        printf("\n");

        return len;
}

int debuglevel = 0;

int dprintf(int lvl, const char *fmt, ...)
{
        va_list ap;
        int ret = 0;

        if (debuglevel >= lvl) {
        va_start(ap, fmt);
        ret = vfprintf(stdout, fmt, ap);
        va_end(ap);
        }

        return ret;
}

void saslerr(int why, const char *what)
{
  fprintf(stderr, "%s: %s", what, sasl_errstring(why, NULL, NULL));
}

void saslfail(int why, const char *what)
{
        saslerr(why, what);
        exit(EX_TEMPFAIL);
}
```

◆ ◆ ◆ 付録 E

SASL リファレンス

この付録では、SASL の関連情報を示します。SASL とは、simple authentication and security layer の略です。

SASL インタフェースの概要

次の表に、いくつかの SASL インタフェースについての簡単な説明を示します。

表 E–1 クライアントとサーバーに共通する SASL 関数

機能	説明
sasl_version	SASL ライブラリのバージョン情報を取得します。
sasl_done	SASL のすべての広域的状態を解放します。
sasl_dispose	接続が完了したときに `sasl_conn_t` を破棄します。
sasl_getprop	プロパティー (ユーザー名、セキュリティー層に関する情報など) を取得します。
sasl_setprop	SASL プロパティーを設定します。
sasl_errdetail	接続に関する最新のエラーから文字列を生成します。
sasl_errstring	SASL エラーコードを文字列に変換します。
sasl_encode	セキュリティー層を使用して送信するデータを符号化します。
sasl_encodev	セキュリティー層を介して伝送するデータブロックを符号化します。入力パラメータとして `iovec *` を使用します。
sasl_listmech	使用できる機構のリストを作成します。
sasl_global_listmech	考えられるすべての機構の配列を戻します。このインタフェースは現在使用されていません。

表 E–1 クライアントとサーバーに共通する SASL 関数 *(続き)*

機能	説明
sasl_seterror	`sasl_errdetail()` によって戻されるエラー文字列を設定します。
sasl_idle	アイドル期間中またはネットワークの往復中に計算を行うように `saslib` を設定します。
sasl_decode	セキュリティー層を使用して受信したデータを復号化します。

表 E–2 クライアント専用の基本的な SASL 関数

機能	説明
sasl_client_init	クライアントプラグインをロードおよび初期化するために最初に 1 度だけ呼び出されます。
sasl_client_new	クライアント接続を初期化します。`sasl_conn_t` コンテキストを設定します。
sasl_client_start	接続に使用する機構を選択します。
sasl_client_step	認証ステップを実行します。

表 E–3 サーバーの基本的な SASL 関数 (クライアントでは任意)

機能	説明
sasl_server_init	サーバープラグインをロードおよび初期化するために最初に 1 度だけ呼び出されます。
sasl_server_new	サーバー接続を初期化します。`sasl_conn_t` コンテキストを設定します。
sasl_server_start	認証の交換を開始します。
sasl_server_step	認証交換ステップを実行します。
sasl_checkpass	平文のパスフレーズを検査します。
sasl_checkapop	APOP のチャレンジおよび応答を検査します。擬似 APOP 機構 (CRAM-MD5 機構に似ている) を使用します。省略可能。このインタフェースは現在使用されていません。
sasl_user_exists	ユーザーが存在するかどうかを確認します。
sasl_setpass	パスワードを変更します。ユーザーエントリを追加します (省略可能)。
sasl_auxprop_request	補助プロパティーを要求します。
sasl_auxprop_getctx	接続用の補助プロパティーコンテキストを取得します。

表 E–4 基本サービスを構成するための SASL 関数

機能	説明
sasl_set_alloc	記憶域割り当て関数を割り当てます。このインタフェースは現在使用されていません。
sasl_set_mutex	相互排他関数を割り当てます。このインタフェースは現在使用されていません。
sasl_client_add_plugin	クライアントプラグインを追加します。
sasl_server_add_plugin	サーバープラグインを追加します。
sasl_canonuser_add_plugin	ユーザーの正規化プラグインを追加します。
sasl_auxprop_add_plugin	補助プロパティープラグインを追加します。

表 E–5 SASL ユーティリティー関数

機能	説明
sasl_decode64	復号化に Base64 を使用します。
sasl_encode64	符号化に Base64 を使用します。
sasl_utf8verify	文字列が有効な UTF-8 かどうかを確認します。
sasl_erasebuffer	セキュリティーに気を配る必要があるバッファーまたはパスワードを消去します。実装では回復されにくい消去ロジックを使用することがあります。

表 E–6 SASL プロパティー関数

機能	説明
`prop_clear()`	値と要求をプロパティーコンテキストから消去します (要求は省略可能)
`prop_dispose()`	プロパティーコンテキストを破棄します
`prop_dup()`	既存の `propctx` の内容を複製して新しい `propctx` を作成します
`prop_erase()`	プロパティーの値を消去します
`prop_format()`	要求されたプロパティー名を文字列に書式設定します
`prop_get()`	コンテキストから `propval` 構造体の配列を戻します
`prop_getnames()`	プロパティー名のリストを指定して、`struct propval` の配列を埋めます
`prop_new()`	プロパティーコンテキストを作成します
`prop_request()`	プロパティー名を要求に追加します

表 E–6 SASLプロパティー関数 (続き)

機能	説明
`prop_set()`	プロパティー値をコンテキストに追加します
`prop_setvals()`	プロパティーの値を設定します
`sasl_auxprop_getctx()`	接続用の補助プロパティーコンテキストを取得します
`sasl_auxprop_request()`	補助プロパティーを要求します

表 E–7 コールバックデータ型

コールバック	説明
sasl_getopt_t	オプション値を取得します。クライアントでもサーバーでも使用されます。
sasl_log_t	メッセージハンドラをロギングします。クライアントでもサーバーでも使用されます。
sasl_getpath_t	機構を探すためのパスを取得します。クライアントでもサーバーでも使用されます。
sasl_verifyfile_t	ファイルがSASLによって使用されるかどうかを確認します。クライアントでもサーバーでも使用されます。
sasl_canon_user_t	ユーザー名の正規化関数。クライアントでもサーバーでも使用されます。
sasl_getsimple_t	ユーザーと言語のリストを取得します。クライアントでのみ使用されます。
sasl_getsecret_t	認証シークレットを取得します。クライアントでのみ使用されます。
sasl_chalprompt_t	チャレンジを表示し、応答を入力するよう求めます。クライアントでのみ使用されます。
sasl_getrealm_t	認証レルムを取得します。クライアントでのみ使用されます。
sasl_authorize_t	ポリシーのコールバックを承認します。サーバーでのみ使用されます。
sasl_server_userdb_checkpass_t	平文のパスワードを検証します。サーバーでのみ使用されます。
sasl_server_userdb_setpass_t	平文のパスワードを設定します。サーバーでのみ使用されます。

表 E–8 SASL インクルードファイル

インクルードファイル	コメント
sasl/saslplug.h	
sasl/sasl.h	プラグインの開発に必要です
sasl/saslutil.h	
sasl/prop.h	

表 E–9 SASL 戻りコード: 一般

戻りコード	説明
SASL_BADMAC	整合性検査が失敗しました
SASL_BADVERS	機構のバージョンが一致しません
SASL_BADPARAM	指定されたパラメータが無効です
SASL_BADPROT	プロトコルが正しくありません。操作を取り消します
SASL_BUFOVER	バッファーがオーバーフローしました
SASL_CONTINUE	認証には別の手順が必要です
SASL_FAIL	一般的なエラー
SASL_NOMECH	機構がサポートされていません
SASL_NOMEM	メモリー不足のため、操作を完了できません
SASL_NOTDONE	交換の後期になるまで情報を要求できません
SASL_NOTINIT	SASL ライブラリが初期化されていません
SASL_OK	結果が正常です
SASL_TRYAGAIN	一時的に発生する障害 (弱い鍵など)

表 E–10 SASL 戻りコード: クライアント専用

機能	説明
SASL_BADSERV	サーバーが相互認証ステップに失敗しました
SASL_INTERACT	ユーザーの操作が必要です
SASL_WRONGMECH	要求された機能が機構でサポートされていません

表 E–11 SASL戻りコード:サーバー専用

機能	説明
SASL_BADAUTH	認証エラー
SASL_BADVERS	バージョンがプラグインと一致していません
SASL_DISABLED	アカウントが無効です
SASL_ENCRYPT	機構を使用するには暗号化が必要です
SASL_EXPIRED	パスフレーズの有効期限が切れているため、リセットする必要があります
SASL_NOAUTHZ	承認エラー
SASL_NOUSER	ユーザーが見つかりません
SASL_NOVERIFY	ユーザーは存在するが、検査方法がありません
SASL_TOOWEAK	このユーザーに対して機構が弱すぎます
SASL_TRANS	使い捨て平文パスワードで、ユーザーに要求された機構が使用できるようになります
SASL_UNAVAIL	リモート認証サーバーが使用できません

表 E–12 SASL戻りコード – パスワード操作

機能	説明
SASL_NOCHANGE	指定された変更は必要ありません
SASL_NOUSERPASS	ユーザーが指定したパスワードは許可されていません
SASL_PWLOCK	パスフレーズがロックされています
SASL_WEAKPASS	セキュリティーポリシーに対してパスフレーズが弱すぎます

◆ ◆ ◆ 付録 F

暗号化プロバイダのパッケージ化と署名

この付録では、Solaris 暗号化プロバイダアプリケーション/モジュールをパッケージ化する方法について説明します。次の項目について説明します。

暗号化プロバイダアプリケーションおよびモジュールのパッケージ化

Solaris オペレーティングシステムでは、アプリケーションソフトウェアは「パッケージ」と呼ばれる単位で配布されます。パッケージとは、ソフトウェア製品の配布とインストールに必要なファイルの集まりです。通常、パッケージは、アプリケーションコードの開発が完了したあとでアプリケーション開発者が設計して作成します。ソフトウェアアプリケーションのパッケージ化の概要については、『アプリケーションパッケージ開発者ガイド』を参照してください。

暗号化プロバイダのパッケージ化には、次の 2 つの要件があります。

- 開発者は、暗号化フレームワークを管理する構成ファイルにアプリケーションを追加するための入力ファイルを用意する必要があります。
- 開発者は、米国政府の輸出法に準拠していることを示す X.509 証明書を用意する必要があります。この証明書は、テストを行う目的で、米国政府の承認を得る前に作成できます。パッケージには承認が必要であり、署名付きプロバイダを出荷しなければなりません。

米国政府の輸出法への準拠

米国政府は、公開された暗号化インタフェース (「crypto-with-a-hole」とも呼ばれる) の輸出を制限しています。この制限のために、プロバイダのすべてのベンダーは米国政府から輸出の承認を得る必要があります。ベンダーは、輸出法に準拠していることを示すために、Sun Microsystems, Inc. が発行する証明書を要求する必要があります。そして、ベンダーは、プロバイダの電子署名を行い、証明書を添付してソフトウェアを出荷します。

輸出承認プロセスでは、暗号化の強度によってソフトウェアを使用できる国が決まります。米国政府では、アメリカ合衆国で製造される暗号化製品に対して 2 つの輸出カテゴリを定義しています。

- リテール版暗号化製品 – リテール版暗号化製品は、セキュリティー上の脅威になると指定された国を除く、すべての国に出荷することが許可されています。
- 非リテール版暗号化製品 – 非リテール版暗号化製品は、国内のみでの使用と米国政府によって認められている国への出荷が許可されています。

プロバイダが非リテール承認を得ている場合は、リテール承認を受けられるようにすることができます。リテール承認を得るには、IPsec などの特定の呼び出し側によるプロバイダの使用を禁止します。Sun では、この場合、制限付きと制限なしの 2 つの証明書を用意します。どちらの証明書を使用するかは、証明書要求プロセス (290 ページの「プロバイダに署名するための証明書を要求するには」) で指定します。また、特別な起動ファイルを作成および署名し、プロバイダとともに出荷する必要があります。293 ページの「リテール版の輸出用の起動ファイルを作成するには」を参照してください。

ユーザーレベルのプロバイダアプリケーションのパッケージ化

Sun 以外で、ユーザーレベルの暗号化プロバイダアプリケーションの開発者は、次の処理を実行します。

1. Sun Microsystems, Inc から証明書を取得します。次に、ライブラリに署名します。290 ページの「プロバイダへの署名の追加」を参照してください。
2. 証明書を添付してパッケージを出荷します。証明書は、`/etc/crypto/certs` ディレクトリに格納する必要があります。
3. `pkcs11conf` クラスを `pkginfo` ファイルの CLASSES 文字列内に追加します。次の行を追加してください。

   ```
   CLASS=none pkcs11conf
   ```
4. 入力ファイル `pkcs11.conf` を `/etc/crypto` ディレクトリに作成します。

ユーザーレベルのプロバイダの入力ファイル名は、pkcs11.conf です。このファイルは、プロバイダへのパスを示します。pkcs11.conf では、エントリに次の構文を使用します。

filename

このエントリは、ファイルへの絶対パス (/opt/lib/$ISA/myProviderApp.so など) です。pkgadd を実行すると、このファイルが構成ファイルに追加されます。パス名の $ISA 表現を書き留めておいてください。$ISA は、必要に応じて、32 ビット版または 64 ビット版のアプリケーションを指します。

5. 次の行をパッケージのプロトタイプファイルに追加します。

```
e pkcs11conf etc/crypto/pkcs11conf 0644 root sys
```

カーネルレベルのプロバイダモジュールのパッケージ化

Sun 以外で、カーネルレベルの暗号化プロバイダモジュールの開発者は、次の処理を実行します。

1. Sun Microsystems, Inc から証明書を取得します。次に、カーネルソフトウェアモジュールまたはデバイスドライバに署名します。290 ページの「プロバイダへの署名の追加」を参照してください。
2. 証明書を添付してパッケージを出荷します。証明書は、/etc/crypto/certs ディレクトリに格納する必要があります。
3. kcfconf クラスを pkginfo ファイルの CLASSES 文字列内に追加します。次の行を追加してください。

```
CLASS=none kcfconf
```

4. 入力ファイル kcf.conf を /etc/crypto ディレクトリに作成します。このファイルは、ソフトウェアとハードウェアのプラグインをカーネル構成ファイルに追加します。
   - プロバイダが暗号化機構を備えたカーネルソフトウェアモジュールである場合は、エントリに次の構文を使用します。

     provider-name:supportedlist=*mech1*,*mech2*,...

     provider-name　カーネルソフトウェアモジュールのベース名

     *mech**　リスト内の暗号化機構の名前

     次のエントリは、カーネルソフトウェアモジュールの例です。

     ```
     des:supportedlist=CKM_DES_CBC,CKM_DES_ECB,CKM_DES_CFB
     ```

   - プロバイダが暗号化機構用のデバイスドライバ (アクセラレータカードなど) である場合は、エントリに次の構文を使用します。

`driver_names=`*devicedriver1*`,`*devicedriver2*`,...`

*devicedriver** 暗号化デバイス用のデバイスドライバの名前

次のエントリは、デバイスドライバの例です。

```
driver_names=dca
```

プロバイダへの署名の追加

この節では、プロバイダに電子署名を追加して、プロバイダが暗号化フレームワーク内で動作できるようにする方法について説明します。また、プロバイダが正しく署名されたかどうかを確認する方法についても説明します。プロバイダは、PKCS #11 ライブラリ、アルゴリズムを実装するロード可能なカーネルモジュール、またはハードウェアアクセラレータ用のデバイスドライバのいずれのオブジェクトでもかまいません。

▼ プロバイダに署名するための証明書を要求するには

一般に、証明書の要求はプロバイダの開発者が行います。しかし、システム管理者がサイトのセキュリティーポリシーの一環として、この要求を処理するように依頼されることもあります。

1 `elfsign request` コマンドを使用して、**Sun** の証明書を要求します。

このコマンドは、証明書の要求に加えて非公開鍵の生成も行います。

```
% elfsign request -k private-keyfile -r certificate-request
```

private-keyfile 非公開鍵の場所へのパス。この鍵は、あとでシステム管理者が Solaris 暗号化フレームワーク用にプロバイダに署名するときに必要となります。このディレクトリはセキュリティー保護しておく必要があります。Sun の証明書が保管されているディレクトリとは別のディレクトリを使用してください。

certificate-request 証明書要求へのパス

次の例は、一般的な要求が Sun にどのように送付されるかを示しています。

```
% elfsign request \
-k /securecrypt/private/MyCompany.private.key \
-r /reqcrypt/MyCompany.certrequest

Enter Company Name / Stock Symbol or some other globally unique identifier.
This will be the prefix of the Certificate DN:MYCORP
```

```
The government of the United States of America restricts the export of
"open cryptographic interfaces", also known as "crypto-with-a-hole".
Due to this restriction, all providers for the Solaris cryptographic
framework must be signed, regardless of the country of origin.

The terms "retail" and "non-retail" refer to export classifications
for products manufactured in the USA.  These terms define the portion of the
world where the product may be shipped.  Roughly speaking, "retail" is
worldwide (minus certain excluded nations) and "non-retail" is domestic
only (plus some highly favored nations).  If your provider is subject to
USA export control, then you must obtain an export approval (classification)
from the government of the USA before exporting your provider.  It is
critical that you specify the obtained (or expected, when used during
development) classification to the following questions so that your provider
will be appropriately signed.

Do you have retail export approval for use without restrictions based
on the caller (for example, IPsec)? [Yes/No] N

If you have non-retail export approval for unrestricted use of your
provider by callers, are you also planning to receive retail approval
restricting which export sensitive callers (for example, IPsec) may
use your provider? [Y/N] Y
```

非公開鍵は、指定したファイル名 (`/etc/crypto/private/MyCompany.private.key` ファイルなど) に格納されます。証明書要求も指定したファイル名 (`/reqcrypt/MyCompany.certrequest` ファイルなど) に格納されます。

2. 証明書要求を **Sun** に送付します。

   証明書要求を次の電子メールアドレスに送信します。 `solaris-crypto-req@sun.com`

   Sun では、証明書要求ファイルから証明書を作成します。作成された証明書のコピーが返送されます。

3. **Sun** から受け取った証明書を `/etc/crypto/certs` ディレクトリに格納します。

   安全のため、非公開鍵と証明書要求は別のディレクトリに格納するようにしてください。

▼ プロバイダに署名するには

一般に、プロバイダへの署名はプロバイダの開発者が行います。しかし、システム管理者がサイトのセキュリティーポリシーの一環として、開発者のバイナリに署名するように依頼されることもあります。

- プロバイダに署名します。`elfsign sign` コマンド、**Sun** から受け取った証明書、および **Sun** の証明書を要求するための非公開鍵を使用します。

  ```
  % elfsign sign -k private-keyfile -c Sun-certificate -e provider-object
  ```

`-k`	Sun に送信された証明書要求の作成に使用された非公開鍵が含まれているファイル。
`-c`	証明書要求によって Sun から発行された証明書へのパス。
`-e`	Solaris 暗号化フレームワーク内で使用するために署名されるプロバイダ (バイナリ) へのパス。

次の例は、プロバイダに署名する方法を示しています。

```
% elfsign sign \
-k /securecrypt/private/MyCompany.private.key \
-c /etc/crypto/certs/MyCompany
-e /path/to/provider.object
```

`elfsign sign` を使用すると、指定された場所にあるオブジェクトが変更されるので注意してください。オブジェクトの未署名版が必要な場合は、`elfsign sign` を適用する前に、オブジェクトを別の場所にコピーする必要があります。

▼ プロバイダが署名されているかどうかを確認するには

1. **Sun** が発行した証明書と署名付きプロバイダへのパスを収集します。
2. `elfsign verify` コマンドを使用して、プロバイダが正しく署名されているかどうかを確認します。

   次の例は、証明書がデフォルトディレクトリ `/etc/crypto/certs/MyCompany` に格納されていると想定した場合の検証を示しています。

   ```
   % elfsign verify \
   -e /path/to/MyProvider.so.1
   elfsign: verification of /path/to/MyProvider.so.1 passed
   ```

   次の例は、証明書がデフォルト以外のディレクトリに格納されている場合を示しています。

   ```
   % elfsign verify \
   -c /path/to/MyCerts  \
   -e /path/to/MyProvider.so.1
   elfsign: verification of /path/to/MyProvider.so.1 passed
   ```

   次の例は、制限付きの証明書で署名されたプロバイダの検証を示しています。

   ```
   % elfsign verify \
   -e /path/to/MyRestrictedProvider.so.1
   elfsign: verification of /path/to/MyRestrictedProvider.so.1 passed, \
   but restricted.
   ```

▼ リテール版の輸出用の起動ファイルを作成するには

この手順は、国内での使用と制限された国外での使用のために同じプロバイダが出荷される場合に役に立ちます。すべての顧客に対して使用制限付き証明書の鍵を使ってプロバイダに署名します。呼び出し側に基づいた制限のないプロバイダを使用する顧客に対しては、IPsec での使用を許可する特別な起動ファイルを作成して組み込みます。起動ファイルは、プロバイダと同じディレクトリに存在させる必要があります。起動ファイルの命名規則では、ドライバの名前に拡張子 .esa を追加します (例: /kernel/drv/vca.esa)。

- プロバイダに署名します。elfsign sign コマンド、**Sun** から受け取った証明書、および **Sun** の証明書を要求するための非公開鍵を使用します。

  % **elfsign sign -a -k** *private-keyfile* **-c** *Sun-certificate* **-e** *provider-object*

  -a　署名付きの ELF 署名用起動 (.esa) ファイルを作成します。このオプションは、暗号化プロバイダが非リテール版の輸出の承認とリテール版の輸出の承認の両方を必要とする場合に使用します。リテール版の承認を行うには、IPsec などの輸出の影響を受けやすい呼び出し側を制限します。このオプションは、プロバイダのバイナリが制限付きの証明書であらかじめ署名されているものとします。

  -k　Sun Microsystems, Inc. に送信された証明書要求の作成に使用された非公開鍵が含まれているファイル。

  -c　証明書要求によって Sun から発行された証明書へのパス。

  -e　Solaris 暗号化フレームワーク内で使用するために署名されるプロバイダ (バイナリ) へのパス。

  次の例は、プロバイダに署名する方法を示しています。

```
% elfsign sign \
-a \
-k /securecrypt/private/MyCompany.private.key \
-c /etc/crypto/certs/MyCompany
-e /path/to/provider.object
```

用語集

GSS-API Generic Security Service Application Programming Interface の略。さまざまなモジュール方式のセキュリティーサービスのサポートを提供するネットワーク層です。GSS-API はセキュリティー認証、整合性、および機密性のサービスを提供します。さらに、セキュリティーに関連して、アプリケーションの移植性を最大限にすることを可能にします。認証、機密性、および整合性の項も参照してください。

MIC メッセージ整合性コード (MIC)の項を参照してください。

MN 機構名 (MN)の項を参照してください。

name 主体の名前。たとえば、`user@machine` などです。GSS-API の名前は `gss_name_t` 構造体を通じて処理されます。このような名前はアプリケーションには不透明です。エクスポート名、機構名 (MN)、名前型、およびprincipalの項も参照してください。

opaque データの値や形式がそれを使用する関数から見えない場合、そのデータに適用されます。たとえば、`gss_init_sec_context()` への *input_token* パラメータはアプリケーションには不透明ですが、GSS-API にとっては重要です。同様に、`gss_wrap()` への *input_message* パラメータは GSS-API には不透明ですが、ラップを行うアプリケーションにとっては重要です。

principal ネットワーク通信に参加する、一意の名前を持つクライアント/ユーザーまたはサーバー/サービスのインスタンス。GSS–API ベースのトランザクションでは主体間の対話が必要となります。次に、主体名の例を示します。

- `user`
- `user@machine`
- `nfs@machine`
- `123.45.678.9`
- `ftp://ftp.company.com`

nameと名前型の項も参照してください。

アクセス制御リスト (ACL) 特定のアクセス権を持つ主体のリストが格納されているファイル。通常、サーバーはアクセス制御リストを調べて、クライアントがサービスを使用するための権限を持っているかどうかを判断します。GSS-API で認証されていても ACL で許可されていなければ、主体はサービスを拒否される可能性があることに注意してください。

委託
: 実際のセキュリティー機構で許可されている場合、主体 (通常はコンテキスト起動側) は、自分の資格とピアとなる主体 (通常はコンテキスト受け入れ側) に「委託」することで、ピア主体をプロキシに指定できます。「委託」された資格を使用すると、ピア主体はオリジナル主体の代わりに要求を行うことができます。たとえば、主体が `rlogin` を使用して、あるマシンから別のマシンにリモートログインする場合などです。

エクスポート名
: `gss_export_name()` によって GSS-API 内部形式から GSS-API エクスポート形式に変換された機構名。エクスポート名は `memcmp()` で GSS-API 以外の文字列形式と比較できます。機構名 (MN)とnameの項も参照してください。

機構名 (MN)
: GSS-API 内部形式名の特別なインスタンス。通常の GSS-API 内部形式名では 1 つの名前に対して複数のインスタンス (それぞれが実際の機構の形式での) を持つことができます。ただし、機構名は特定の機構に一意です。機構名は `gss_canonicalize_name()` で生成されます。

機密性
: データを暗号化するセキュリティーサービス。機密性には整合性と認証のサービスも含まれます。認証、整合性、サービスも参照してください。

クライアント
: 狭義では、`rlogin` を使用するアプリケーションなど、ユーザーの代わりにネットワークサービスを使用するプロセスを指します。サーバー自身が他のサーバーやサービスのクライアントになる場合もあります。広義では、サービスを使用する主体を指します。

誤順序の検出
: 多くのセキュリティー機構では、メッセージストリーム中のメッセージが不適切な順序で受信されたことを検出できます。メッセージの誤順序の検出は、(利用できる場合は) コンテキスト確立時に要求する必要があります。

コンシューマ
: システムサービスを使用するアプリケーション、ライブラリ、またはカーネルモジュール。

コンテキスト
: 2 つのアプリケーション間の信用の状態。2 つのピア間でコンテキストが正常に確立されると、コンテキスト受け入れ側はコンテキスト起動側が本当に主張しているとおりのアプリケーションであることを認識して、コンテキスト受け入れ側に送信されたメッセージを検証および復号化できます。コンテキストに相互認証が含まれている場合、起動側は受け入れ側の ID が有効であると認識して、受け入れ側から送信されたメッセージを検証および復号化できます。

コンテキストレベルトークン
: トークンの項を参照してください。

サーバー
: ネットワーククライアントにリソースを提供する主体。たとえば、`rlogin` を使用して `boston.eng.acme.com` というマシンにログインすると、そのマシンは `rlogin` サービスを提供するサーバーになります。

サービス
: 1. (ネットワークサービスと同意)。ネットワーククライアントに提供されるリソース。複数のサーバーによって提供されることもあります。たとえば、`rlogin` を使用して `boston.eng.acme.com` というマシンにログインすると、そのマシンは `rlogin` サービスを提供するサーバーになります。

  2. 「セキュリティーサービス」は整合性または機密性のサービスであり、認証以上の保護レベルを提供します。認証、整合性、および機密性の項も参照してください。

資格
: 主体を識別する情報パッケージと主体の識別情報。資格は、主体がだれであるか、そして多くの場合、主体がどのような特権を持っているかを示します。資格はセキュリティー機構によって生成されます。

資格キャッシュ
: 指定された機構によって保存された資格を保持するための保存領域 (通常はファイル)。

承認
: 主体がサービスを使用できるかどうか、主体がどのオブジェクトにアクセスできるか、および、各オブジェクトにどのようなアクセスの種類が許可されているかを決定するプロセスです。

整合性
: ユーザー認証に加えて、転送されたデータの有効性を暗号タグで証明するセキュリティーサービス。認証、機密性、およびメッセージ整合性コード (MIC)の項も参照してください。

セキュリティーサービス
: サービスを参照してください。

セキュリティーフレーバ
: フレーバを参照してください。

セキュリティーメカニズム
: メカニズムを参照してください。

相互認証
: コンテキストが確立されるとき、コンテキスト起動側は自分自身をコンテキスト受け入れ側に認証する必要があります。場合によっては、コンテキスト起動側が受け入れ側の認証を要求することもあります。受け入れ側が自己認証を行なった場合、両者は相互認証されていると言います。

データ型
: データの形式。たとえば、`int`、`string`、`gss_name_t` 構造体、`gss_OID_set` 構造体などです。

データリプレイ
: メッセージストリーム中の単一のメッセージが複数回受信された場合。多くのセキュリティー機構でデータリプレイの検出をサポートしています。リプレイの検出は、(利用できる場合は) コンテキスト確立時に要求する必要があります。

トークン
: GSS-API 構造体 `gss_buffer_t` の形式であるデータパケット。トークンは、ピアとなるアプリケーションへの転送用に、GSS-API 関数で生成されます。

  トークンには 2 種類あります。コンテキストレベルトークンには、セキュリティーコンテキストを確立または管理するために使用される情報が格納されます。たとえば、`gss_init_sec_context()` は、コンテキストの受け入れ側に送信するための、コンテキスト起動側の資格ハンドル、ターゲットマシンの名前、要求されるさまざまなサービスのフラグなどの項目をトークンに格納します。

  メッセージトークン (メッセージ毎トークンやメッセージレベルトークンとも呼ぶ) には、ピアとなるアプリケーションに送信されるメッセージから GSS-API 関数によって生成された情報が格納されます。 たとえば、`gss_get_mic()` は、指定されたメッセージから識別用の暗号タグを生成し、ピアに送信されるトークンに (メッセージと一緒に) 格納します。技術的には、トークンはメッセージとは別であると考えられています。このため、`gss_wrap()` は *output_token* ではなく *output_message* を生成すると言われます。

メッセージの項も参照してください。

名前型
: 名前の形式。名前型は gss_OID 型として格納され、名前に使用されている形式を示します。たとえば、名前 user@machine の名前型は、GSS_C_NT_HOSTBASED_SERVICE になります。エクスポート名、機構名 (MN)、およびnameの項も参照してください。

認証
: 要求された主体の ID を確認するセキュリティーサービス。

プライバシ
: 機密性の項を参照してください。

フレーバ
: 従来、フレーバは認証の種類 (AUTH_UNIX、AUTH_DES、AUTH_KERB など) を示していたため、「セキュリティーフレーバ」と「認証フレーバ」は同じ意味です。RPCSEC_GSS もセキュリティーフレーバですが、これは認証に加えて、整合性と機密性のサービスも提供します。

プロバイダ
: サービスをコンシューマに提供するアプリケーション、ライブラリ、またはカーネルモジュール。

保護品質 (QOP)
: QOP は Quality of Protection の略で、整合性や機密性のサービスと一緒に使用される暗号化アルゴリズムを選択するときに使用されるパラメータです。整合性と一緒に使用する場合、QOP はメッセージ整合性コード (MIC) を生成するアルゴリズムを指定します。機密性と一緒に使用する場合、QOP は MIC の生成とメッセージの暗号化の両方に対するアルゴリズムを指定します。

ホスト
: ネットワークを通じてアクセス可能なマシン。

メカニズム
: データの認証や機密性を実現するための暗号化技術を指定するソフトウェアパッケージ。たとえば、Kerberos v5 や Diffie-Hellman 公開鍵などです。

メッセージ
: GSS-API ベースのアプリケーションからピアとなるアプリケーションに送信される gss_buffer_t オブジェクト形式のデータ。たとえば、「ls」はリモートの ftp サーバーにメッセージとして送信されます。

  メッセージには、ユーザーが提供するデータ以外の情報が格納されることもあります。たとえば、gss_wrap() はラップされていないメッセージを受け取り、そのメッセージを送信用にラップします。このとき、ラップされたメッセージには、オリジナル (ユーザーが提供した) メッセージとともにその MIC が格納されます。メッセージを含まない、GSS-API が生成した情報は「トークン」と呼ばれます。トークンの項を参照してください。

メッセージ整合性コード (MIC)
: データの有効性を保証するために、転送されるデータに添付される暗号タグ。データ受信側は別の MIC を生成し、送信された MIC と比較します。両者が同じ場合、メッセージは有効です。gss_get_mic() で生成される MIC などはアプリケーションからも見えますが、gss_wrap() や gss_init_sec_context() で生成される MIC などはアプリケーションからは見えません。

メッセージ毎トークン
: トークンの項を参照してください。

メッセージレベルトークン
: トークンの項を参照してください。

リプレイの検出

多くのセキュリティー機構は、メッセージストリーム中のメッセージが不正に繰り返されたことを検出できます。メッセージリプレイの検出は、(利用できる場合は) コンテキスト確立時に要求する必要があります。

索引

G

Q

R

S

T

U

X

あ

ら

り

れ