

# Tru64 UNIX

---

## セキュリティ・プログラミング・ガイド

Part Number: AA-RTFKA-TE

**2002 年 11 月**

ソフトウェア・バージョン: Tru64 UNIX Version 5.1B 以上

本書は、Tru64 UNIX のセキュリティ機能の使用に対応したプログラムの記述方法を説明します。

---

日本ヒューレット・パッカード株式会社

---

© 2002 日本ヒューレット・パッカード株式会社

本書の著作権は日本ヒューレット・パッカード株式会社が保有しており、本書中の解説および図、表は日本ヒューレット・パッカードの文書による許可なしに、その全体または一部を、いかなる場合にも再版あるいは複製することを禁じます。

日本ヒューレット・パッカードは、弊社または弊社の指定する会社から納入された機器以外の機器で対象ソフトウェアを使用した場合、その性能あるいは信頼性について一切責任を負いかねます。

本書に記載されている事項は、予告なく変更されることがありますので、あらかじめご承知おきください。万一、本書の記述に誤りがあった場合でも、弊社は一切その責任を負いかねます。

本書で解説するソフトウェア(対象ソフトウェア)は、所定のライセンス契約が締結された場合に限り、その使用あるいは複製が許可されます。

COMPAQ, Compaq ロゴ, Digital ロゴは U.S. Patent and Trademark Office に登録されています。Alpha, AlphaServer, NonStop, TruCluster, および Tru64 は米国 Compaq Computer Corporation の商標です。

Microsoft, Windows および Windows NT は米国 Microsoft 社の登録商標です。Intel は米国 Intel 社の登録商標です。Motif, OSF/1, UNIX, The Open Group および X/Open は、The Open Group の米国ならびに他の国における商標です。

このドキュメントに記載されているその他の会社名および製品名は、各社の商標または登録商標です。

原典: Security Programming (AA-RSFDA-TE)  
© 2002 Hewlett-Packard Company

---

## 目次

### まえがき

## 1 プログラマ向けセキュリティ機能の概要

1.1	セキュリティ・プログラミングの概要 .....	1-2
1.1.1	TCB ファイルの保護 .....	1-3
1.2	ライブラリとヘッダ・ファイル .....	1-5
1.3	標準のトラステッド・システム・ディレクトリ .....	1-6
1.4	セキュリティに関連するシステム・コールとライブラリ・ルーチン .....	1-7
1.4.1	システム・コール .....	1-7
1.4.2	ライブラリ・ルーチン .....	1-8

## 2 トラステッド・プログラミング技法

2.1	SUID プログラムと SGID プログラムの書き方 .....	2-1
2.2	エラー処理 .....	2-2
2.3	ファイルの保護 .....	2-4
2.4	安全な探索パスの指定 .....	2-5
2.5	シグナルへの応答 .....	2-5
2.6	子プロセスでのオープン・ファイル記述子の使用 .....	2-6
2.7	X 環境でのセキュリティ上の考慮事項 .....	2-7
2.7.1	キーボード入力の保護 .....	2-7
2.7.2	キーボードおよびマウスのイベントのブロック .....	2-8
2.7.3	デバイス関連イベントの保護 .....	2-9
2.8	シェル・スクリプトの保護 .....	2-10

### 3 認証データベース

3.1	認証データベースの概要 .....	3-1
3.1.1	デバイス割り当てデータベース (devassign) .....	3-2
3.1.2	ファイル制御データベース .....	3-2
3.1.3	システム省略時設定データベース (default) .....	3-3
3.1.4	エンハンスト (保護) パスワード・データベース .....	3-4
3.1.5	端末制御データベース .....	3-5
3.2	認証データベースの構成要素 .....	3-6
3.2.1	データベースの形式 .....	3-6
3.2.2	データベースの読み取りと書き込み .....	3-8
3.2.2.1	バッファ管理 .....	3-8
3.2.2.2	名前または ID でのエントリの読み取り .....	3-10
3.2.2.3	エントリの順次読み取り .....	3-10
3.2.2.4	システム省略時設定の使用 .....	3-11
3.2.2.5	エントリの書き込み .....	3-12
3.3	認証データベースのアクセス .....	3-13

### 4 識別と認証

4.1	監査 ID .....	4-1
4.2	識別サポート・ライブラリ .....	4-2
4.3	デーモンの使用 .....	4-2
4.4	エンハンスト (保護) パスワード・データベースの使用 .....	4-3
4.4.1	例:パスワードの有効期限出力プログラム .....	4-5

### 5 監査レコードの生成

5.1	監査レコードの概要 .....	5-1
5.2	監査イベント .....	5-2
5.3	監査レコードとトークン .....	5-2

5.3.1	パブリック・トークン .....	5-3
5.3.2	プライベート・トークン .....	5-4
5.4	監査フラグとマスク .....	5-4
5.5	現プロセスのシステム・コール監査の無効化 .....	5-6
5.6	現プロセスのシステム・コールの監査の変更 .....	5-6
5.7	アプリケーション固有の監査レコード .....	5-7
5.8	サイト定義イベント .....	5-8
5.8.1	サンプルの site_events ファイル .....	5-9
5.8.2	例: サイト定義監査イベントの監査レコードの生成 .....	5-9
5.9	独自の監査ログの作成 .....	5-10
5.10	監査ログの解析 .....	5-10
5.10.1	監査ログのフォーマットの概要と共通タブルのリスト ....	5-11
5.10.2	トークンおよびタブルのバイト記述 .....	5-12
5.10.3	タブルの解析 .....	5-16

## 6 SIA インタフェースの使用

6.1	SIA の概要 .....	6-1
6.2	SIA のアーキテクチャ .....	6-6
6.2.1	ライブラリ .....	6-7
6.2.2	ヘッダ・ファイル .....	6-7
6.3	SIA システムの初期設定 .....	6-7
6.4	SIAENTITY 構造体 .....	6-8
6.5	SIA パラメータの収集 .....	6-8
6.6	状態の保守 .....	6-10
6.7	SIA のリターン値 .....	6-11
6.8	SIA のデバッグとログ .....	6-11
6.9	SIA のセキュリティ・メカニズムの統合 .....	6-12
6.10	SIA セッション処理 .....	6-14
6.10.1	セッションの初期化 .....	6-18

6.10.2	セッションの認証 .....	6-19
6.10.3	セッションの確立 .....	6-20
6.10.4	セッションの起動 .....	6-20
6.10.5	セッションの解放 .....	6-20
6.10.6	特定のセッション処理 .....	6-20
6.10.6.1	login プロセス .....	6-21
6.10.6.2	rshd プロセス .....	6-21
6.10.6.3	rlogind プロセス .....	6-21
6.11	セキュア情報の変更 .....	6-21
6.11.1	ユーザのパスワードの変更 .....	6-21
6.11.2	ユーザの finger 情報の変更 .....	6-22
6.11.3	ユーザのシェルの変更 .....	6-22
6.12	セキュリティ情報へのアクセス .....	6-22
6.12.1	/etc/passwd 情報へのアクセス .....	6-23
6.12.2	/etc/group 情報へのアクセス .....	6-24
6.13	セッション・パラメータの収集 .....	6-24
6.14	SIA 製品のパッケージ .....	6-24
6.15	セキュリティ・メカニズムに依存するインタフェース .....	6-25
6.16	シングルユーザ・モード .....	6-27
6.17	SIA ルーチンのシンボル優先使用 .....	6-27
6.17.1	シンボル優先使用の問題の概要 .....	6-27
6.17.2	Tru64 UNIX での解決方法 .....	6-28
6.17.3	シングルユーザ環境の置き換え .....	6-29

## 7 ACL のプログラミング

7.1	ACL の概要 .....	7-1
7.2	ACL のデータ表現 .....	7-2
7.2.1	内部データ表現 .....	7-2
7.2.1.1	typedef struct acl *acl_t; .....	7-3

7.2.1.2	typedef struct acl_entry *acl_entry_t; .....	7-3
7.2.1.3	typedef uint_t acl_type_t; .....	7-3
7.2.1.4	typedef uint_t acl_tag_t; .....	7-4
7.2.1.5	typedef uint_t acl_perm_t; .....	7-4
7.2.1.6	typedef acl_perm_t *acl_permset_t; .....	7-4
7.2.1.7	連続内部表現 ACL .....	7-5
7.2.2	外部表現 .....	7-5
7.3	ACL ライブラリ・ルーチン .....	7-5
7.4	ACL の規則 .....	7-8
7.4.1	オブジェクトの作成 .....	7-9
7.4.2	ACL の複製 .....	7-9
7.4.3	ACL の有効性 .....	7-9
7.5	ACL の作成例 .....	7-10
7.6	ACL の継承例 .....	7-14

## 8 GSS-API

8.1	GSS-API の概要 .....	8-1
8.1.1	GSS-API の前提条件 .....	8-3
8.1.2	詳細情報 .....	8-4
8.2	Application Security SDK .....	8-5
8.3	Application Security SDK 関数 .....	8-6
8.3.1	名前管理関数 .....	8-6
8.3.1.1	省略時の名前と構文 .....	8-9
8.3.2	信任状管理関数 .....	8-10
8.3.2.1	初回信任状の取得 .....	8-13
8.3.2.1.1	開始側のアプリケーション .....	8-13
8.3.2.1.2	受け入れ側のアプリケーション .....	8-13
8.3.2.1.3	DES3 .....	8-13
8.3.2.2	信任状の属性 .....	8-14
8.3.2.3	信任状の格納場所 .....	8-15

8.3.2.4	信任状関連リソースの管理 .....	8-15
8.3.3	セキュリティ・コンテキスト管理関数 .....	8-16
8.3.3.1	メカニズムの識別 .....	8-17
8.3.3.2	トークンの交換 .....	8-17
8.3.3.3	オプションのセキュリティ手段 .....	8-18
8.3.3.3.1	チャンネル・バインディング .....	8-18
8.3.3.3.2	機密性と完全性 .....	8-20
8.3.3.3.3	リプレイ検出 .....	8-20
8.3.3.3.4	順序外メッセージ検出 .....	8-21
8.3.3.3.5	相互認証 .....	8-21
8.3.3.3.6	暗号化タイプ：DES と DES3 .....	8-21
8.3.3.3.7	信任状の委任 .....	8-21
8.3.3.4	対象とするセキュリティ手段の指定 .....	8-21
8.3.4	メッセージ関数 .....	8-22
8.3.4.1	Quality of Protection .....	8-23
8.3.5	その他のサポート関数 .....	8-24
8.3.5.1	OID と OID セット .....	8-26
8.3.5.1.1	OSI .....	8-26
8.3.5.1.2	ASN.1 .....	8-26
8.3.5.1.3	オブジェクト識別子 .....	8-26
8.3.5.1.4	OID セット .....	8-27
8.3.6	V1 準拠関数 .....	8-27
8.4	ベスト・プラクティス .....	8-28
8.4.1	マルチ・スレッディング .....	8-28
8.4.2	キャッシュ管理 .....	8-29
8.4.3	暗号化タイプ .....	8-29
8.4.4	セキュリティ・コンテキストのエクスポート .....	8-29
8.4.5	GSS と Kerberos 5 による鍵管理 .....	8-29
8.4.6	マルチ・スレッド関数 .....	8-31



8.4.7	相互認証 .....	8-32
8.4.8	パスワードの保護 .....	8-32
8.4.9	リプレイの防止 .....	8-32
8.4.10	信任状のリフレッシュ .....	8-33
8.4.11	リソース管理 .....	8-33
8.4.12	サービス鍵テーブル・ファイル .....	8-33
8.4.13	チケット属性 .....	8-34
8.4.13.1	転送可能チケット .....	8-34
8.4.13.2	事前認証 .....	8-34
8.4.13.3	チケットの存続期間 .....	8-35
8.4.13.4	チケットの延長期限 .....	8-35
8.4.13.4.1	存続期間と延長の設定に関する一般的な規則 .....	8-35
8.5	移植性のあるアプリケーションの構築 .....	8-36
8.5.1	表示可能な名前の使用と名前の比較 .....	8-36
8.5.2	メカニズムの指定 .....	8-37
8.5.3	Quality of Protection (QOP) の指定 .....	8-37
8.5.4	省略時の名前 .....	8-38
8.6	クリック・リファレンス .....	8-39
8.6.1	リファレンス・ページの表記規則 .....	8-42
8.7	定数 .....	8-44
8.8	データ構造体 .....	8-47
8.8.1	gss_channel_bindings_t .....	8-47
8.8.2	gss_buffer_t .....	8-48
8.8.3	csf_gss_opts_t .....	8-48
8.9	リターン値 .....	8-49
8.9.1	定義されているステータス・コード .....	8-49
8.9.2	エラー処理マクロ .....	8-49
8.9.2.1	GSS_ERROR() .....	8-50
8.9.2.2	GSS_CALLING_ERROR() .....	8-50
8.9.2.3	GSS_ROUTINE_ERROR() .....	8-51

8.9.2.4	GSS_SUPPLEMENTARY_INFO() .....	8-52
8.9.3	メジャー・ステータス .....	8-53
8.9.4	マイナー・ステータス .....	8-55
8.9.5	Kerberos 固有のコード .....	8-57
 <b>A コーディング例</b>		
A.1	再認証プログラムのソース・コード (sia-reauth.c) .....	A-1
A.2	スーパーユーザ認証プログラムのソース・コード (sia-suauth.c) .....	A-2
 <b>B 監査可能イベントとエイリアス</b>		
B.1	省略時の監査可能イベント・ファイル .....	B-1
B.2	イベント・エイリアス・ファイルのサンプル .....	B-5
 <b>C GSS-API チュートリアル</b>		
C.1	セキュリティの基礎 .....	C-1
C.1.1	基本概念 .....	C-1
C.1.2	Kerberos セキュリティ・モデル .....	C-2
C.1.2.1	定義 .....	C-2
C.1.2.2	概念と処理手順 .....	C-3
C.1.2.2.1	共有秘密 .....	C-4
C.1.2.2.2	信用できる第三者による調停 .....	C-4
C.1.2.2.3	Kerberos ネットワーク .....	C-4
C.1.2.2.4	認証までの 3 つの段階 .....	C-4
C.1.2.2.5	認証サービス・メッセージの交換 .....	C-5
C.1.2.2.6	チケット・グランティング・サービス・メッセージの交換 .....	C-6
C.1.2.2.7	アプリケーション間のメッセージの交換 .....	C-6
C.1.2.3	信任状の属性 .....	C-7
C.2	はじめに .....	C-7
C.3	基本的な GSS-API 関数の使用 .....	C-8

C.4	手順 1 : 名前の取得 .....	C-10
C.5	手順 2 : 信任状の取得 .....	C-12
C.6	手順 3 : セキュリティ・コンテキストの確立 .....	C-16
C.7	手順 4 : メッセージの交換 .....	C-24
C.7.1	gss_get_mic() と gss_verify_mic() の使用 .....	C-27
C.7.2	gss_wrap() と gss_unwrap() の使用 .....	C-29
C.8	手順 5 : セキュリティ・コンテキストの終了 .....	C-33
C.9	高度な概念 .....	C-34
C.9.1	初回信任状の取得 .....	C-34
C.9.2	必要な時刻同期 .....	C-38
C.9.3	DES3 暗号化の使用 .....	C-38
C.10	GSS-API 関数のステータス・コード .....	C-39
C.10.1	マイナー・ステータス・コード .....	C-41
C.11	サンプル・プログラム .....	C-42
C.11.1	サンプル・プログラムのビルド .....	C-46
C.11.2	サンプル・プログラムの実行 .....	C-46
C.11.2.1	前提条件 .....	C-46
C.11.2.2	サンプル・プログラムの設定 .....	C-47
C.11.2.3	サーバのコマンド行スイッチ (省略可能) .....	C-48
C.11.2.4	クライアントのコマンド行スイッチ (省略可能) .....	C-48
C.11.3	サンプル・プログラムの出力 .....	C-49
C.11.4	トラブルシューティングのガイドライン .....	C-51

## 索引

### 例

4-1	パスワードの有効期限出力プログラム .....	4-5
6-1	SIAENTITY 構造体 .....	6-8
6-2	sia.h でのパラメータ収集インタフェースの定義 .....	6-9

6-3	一般的な /var/adm/sialog ファイル .....	6-12
6-4	login コマンドのセッション処理 .....	6-16
6-5	シングルユーザ環境のシンボル優先使用 .....	6-29
8-1	文字列の入った構造体を指す定数 .....	8-27
8-2	文字列を指す定数 .....	8-27
A-1	再認証プログラム .....	A-1
A-2	スーパーユーザ認証プログラム .....	A-2

## 図

6-1	SIA の階層構造 .....	6-2
6-2	SIA セッション処理 .....	6-14

## 表

1-1	標準のトラステッド・システム・ディレクトリ .....	1-6
1-2	セキュリティに関連するシステム・コール .....	1-7
1-3	セキュリティに関連するライブラリ・ルーチン .....	1-8
5-1	大半の監査レコードに共通する省略時のタプル .....	5-11
5-2	トークンおよびタプルのバイト記述 .....	5-12
6-1	セキュリティを扱うオペレーティング・システム・コマンド	6-2
6-2	SIA のメカニズム非依存ルーチン .....	6-3
6-3	SIA のメカニズム依存ルーチン .....	6-4
7-1	ACL エントリの外部表現 .....	7-5

---

## まえがき

本書は、HP Tru64 UNIX のセキュリティ機能を利用したプログラムの作成方法について説明します。

### 本書の対象読者

本書は、UNIX システムでセキュリティ関連のプログラムを作成、変更しているプログラマの方、および C 言語のプログラミングに精通したプログラマの方を対象にしています。

### 本書の構成

本書の各章の内容は以下のとおりです。

- |       |   |
|-------|---|
| 第 1 章 | 本書の例で共通に使用される事項を説明するとともに、トラステッド・コンピューティング・ベースに関する情報を示します。   |
| 第 2 章 | 直接実行されるコマンドあるいはデーモンのどちらとしてプログラムを作成するかなど、トラステッド・プログラムの設計での個々の技法を説明します。   |
| 第 3 章 | 認証データベースの構造とそれを問い合わせる方法について説明します。   |
| 第 4 章 | オペレーティング・システムの各種のユーザ ID およびグループ ID についてと、それらの使用方法 (特に従来の UNIX システムにはない監査 ID の使用方法) を説明します。また、エンハンスド (保護) パスワード・データベースの内容についても説明します。 |
| 第 5 章 | トラステッド・プログラムが監査ログ内にエントリを記録する時期およびエントリを記録するメカニズムに関するガイドラインを示します。   |
| 第 6 章 | セキュリティ統合アーキテクチャ (SIA) のプログラミング・インタフェースについて説明しています。  |
| 第 7 章 | Tru64 UNIX で実行するアプリケーションでのアクセス制御リスト (ACL) の使用について説明します。   |
| 第 8 章 | GSS-API の標準およびセキュリティ基盤、および GSS Application Security SDK の関数呼び出しと、これらを使用するためのベスト・プラクティスと移植性の問題を取り上げます。                               |
| 付録 A  | トラステッド Tru64 UNIX システムのコーディング例を示します。  |

- 付録 B 省略時の監査可能イベント (/etc/sec/audit\_events) および省略時の監査イベント・エイリアス (/etc/sec/event\_aliases) のファイルを示します。
- 付録 C GSS-API を使用してアプリケーションをセキュアにする方法を、C 言語によるサンプル・コードを用いて説明します。Application Security SDK に含まれているサンプル・プログラムについても説明します。

## 参考資料

以下のドキュメントは、本書の各章の情報を補足する重要事項を取り上げています。

- 『セキュリティ管理ガイド』では、Tru64 UNIX の一般的な管理操作を実行する方法について説明しています。
- 『リリース・ノート』には、マニュアルにはない、セキュリティに関する重要な情報が含まれている場合があります。

## 表記法

本書では以下の表記法を使用します。

<code>\</code>	例中の行末のバックスラッシュは、継続を表します。
<code>#</code>	番号記号は root として Tru64 UNIX システムにログインした場合のシステム・プロンプトを表します。
<code>net stop</code>	太字 (ボールド体) はユーザが入力する文字を示します。
<code>&gt;&gt;&gt;</code>	コンソール・サブシステムのプロンプトです。
<code>file</code>	イタリック体 (斜体) は、変数値、プレースホルダ、および関数の引数名を示します。
<code>[   ]</code> <code>{   }</code>	構文定義では、大カッコはオプションの項目を示し、中カッコは必須項目を示します。大カッコまたは中カッコの中の項目を縦線で区切っている場合は、そこに併記されている項目の中から 1 つの項目を選択することを示します。
<code>...</code>	構文定義では、水平の反復記号は、前の項目を 1 回以上繰り返して使用できることを示します。

cat(1)

リファレンス・ページの参照には、該当するセクション番号をカッコ内に示します。たとえば、cat(1) は、cat コマンドについての情報が、リファレンス・ページのセクション 1 に記載されていることを示します。

[Ctrl/x]

この記号は、スラッシュの前に指定されているキーを押しながら、スラッシュの後のキーまたはマウス・ボタンを押すことを示します。例中では、このようなキーの組み合わせは、四角あるいは大カッコで囲まれて示されます (たとえば、[Ctrl/C])。





## プログラマ向けセキュリティ機能の概要

この章では、トラステッド Tru64 UNIX システム上でのトラステッド・アプリケーションの実行に関連する事項について説明しています。ライブラリ、ヘッダ・ファイル、標準のトラステッド・システム・ディレクトリ、およびトラステッド・コンピューティング・ベース (TCB) について記述しています。この章およびこの章以降では、C プログラムの一部や全体を使用して、基本的な考え方を説明します。変更せずに使用できるプログラムも記載されていますが、これらのルーチンを集めても、トラステッド・プログラムを構築することはできません。

本書で説明するプログラミング技法のほかに、次の技法も使用できます。

- Common Data Security Architecture (CDSA)

CDSA はマルチプラットフォーム、業界標準のセキュリティ構造です。標準規格に基づく、安定したプログラミング・インタフェースを提供します。アプリケーションはこのインタフェースを使用して、オペレーティング・システムのセキュリティ・サービスにアクセスできます。これによって開発者は、クロスプラットフォームの、セキュリティに対応したアプリケーションを作成できます。アプリケーションは、動的拡張性のあるアプリケーション・プログラミング・インタフェース (API) を通じて、暗号化や他の公開鍵処理などのセキュリティ・サービスをリクエストします。このリクエストは、一連のプラグイン・セキュリティ・サービス・モジュール (SPI) によって遂行されます。SPI は、ビジネス要件と技術の発展に従って補ったり、変更したりできます。

CDSA についての詳細は、`cdsa(3)` を参照してください。

- Secure Socket Layer (SSL)

SSL は、インターネット上でのメッセージの送受信のセキュリティを管理するための一般的なプロトコルです。最近では、SSL をベースにした Transport Layer Security (TLS) が SSL に取って代わっています。SSL は Hypertext Transfer Protocol (HTTP) レイヤと Transport Control Protocol (TCP) レイヤの間にあるプログラム・レイヤを使用します。

提供されている OpenSSL ライブラリは、Secure Sockets Layer (SSL v2/v3) および Transport Layer Security (TLS v1) プロトコルを実装しています。

SSL についての詳細は、`ssl(3)`を参照してください。

この章では、次の内容について説明しています。

- セキュリティ・プログラミングの概要
- ライブラリとヘッダ・ファイル
- 標準のトラステッド・システム・ディレクトリ
- セキュリティに関連するシステム・コールとライブラリ・ルーチン

この章およびこの章以降では、C プログラムの一部や全体を使用して、基本的な考え方を説明します。変更せずに使用できるプログラムも記載されていますが、これらのルーチンを集めても、トラステッド・プログラムを構築することはできません。

## 1.1 セキュリティ・プログラミングの概要

トラステッド・コンピューティング・ベース (TCB) を誤って変更しないように、保護しなければなりません。そのためには、どのプログラムとデータ・ファイルが TCB の一部なのかをまず定義します。以下に、TCB の構成要素について説明します。

- **トラステッド・プログラム**：セキュリティ・ルールを破壊できる立場にあるプログラムは、トラステッド・プログラムでなければなりません。このようなプログラムには、直接セキュリティ上の決定を行うプログラムと、決定は行わないがプログラムに誤りまたは不正なコードがある場合にセキュリティ・ルールを破壊する可能性があるプログラムがあります。ユーザ ID に root (SUID) が設定されているプログラム・ファイルの場合は、トラステッド・プログラムであると考えられます。
- **間接プログラム**：他のトラステッド・プログラムから起動されたり、他のトラステッド・プログラムと連携して動作することによって、セキュリティ上の決定に利用されるプログラムは、トラステッド・プログラムです。他のトラステッド・プログラムが依存しているデータ・ファイルやその他のオブジェクトを変更するプログラムも、トラステッド・プログラムです。

- プログラム・ファイル：トラステッド・プログラムが格納されている実行可能ファイルは、TCBの一部とみなされます。
- オブジェクト・コードとライブラリ：静的リンクか動的リンクにかかわらず、トラステッド・プログラムに組み込まれるオブジェクト (バイナリ) コード・モジュールとそれらのファイルはすべて、TCBの一部です。これには、トラステッド・プログラムで頻繁に使われる標準 C ライブラリ・ルーチンとインタフェースがあります。
- データ・ファイル：TCB には、セキュリティ上の決定を行うためにトラステッド・プログラムで使用するデータが格納されたファイル (たとえば、`ttys` データベース) が含まれます。
- シェル・スクリプト：シェル・スクリプトはデータ・ファイルで、シェル・プログラムがこのファイル内のシェル・コマンドを解釈し、実行します。シェル・スクリプトが、トラステッド・プログラムに代わって機能を実行するか、トラステッド・システムの正常な動作に欠かせない場合、そのシェル・スクリプトはTCBの一部とみなされます。スクリプトの削除や移動により、システムが誤って動作する (たとえば、一部の `rc` スタートアップ・スクリプトを削除する) か、セキュリティが侵害される可能性がある (異なる `cron` スタートアップ・ファイルをインストールする) 場合は、そのシェル・スクリプトがセキュリティに関連すると判断することができます。シェル・スクリプト・ファイルは、オブジェクト・コード・プログラム・ファイルと同じように、慎重に保護する必要があります。シェル・スクリプトを実行するには、そのスクリプトが読み取り可能でなければなりません。
- 親ディレクトリ：TCB ファイルの親ディレクトリはすべて TCB の一部とみなし、同様に保護する必要があります。悪意を持ったユーザがこれらのディレクトリでリンクを削除したり再定義することができると、このユーザが新しくにせのファイルを作成することができます。これにより、トラステッド・プログラムが、誤ったセキュリティ上の決定を行う可能性があります。

### 1.1.1 TCB ファイルの保護

以下の各メカニズムは、TCB のファイルとディレクトリを保護する方法を提供しています。

- *Discretionary Access Control (DAC)*: 任意アクセス制御 (所有者, グループ, モード・ビット, およびアクセス制御リスト (ACL)) は, TCB ファイルに対する, 最も重要な保護です。信頼できないユーザとグループに対しては, ファイルの読み取りを許可しても, ファイルの変更を行えないようにしなければなりません。このためには, 疑似ユーザと疑似グループを作成するのが一般的です。

既存のプログラムは, ファイルを複製するときにモード・ビットだけをコピーし, 誤って ACL を削除してしまうことがあります。これにより, ACL による保護が行われなくなります。other::--- および group::--- のような制限された従来の許可コードを使用してから, ユーザ・エントリによって個々のユーザへアクセス権を与えてください。この方法を使うと, ACL が失われても, 意図していないアクセスは許可されません。ACL を使用したプログラミングについては, 第 7 章を参照してください。

- 読み取り専用ファイル・システム: 読み取りだけが必要なファイルを, すべて別のファイル・システムに置き, そのファイル・システムを読み取り専用としてマウントできます。これにより, どのプログラムも特権に関係なく, これらのファイルを変更することができなくなります (ファイル・システムを再マウントした場合を除く)。ファイルを変更する必要がある場合には, ファイル・システムを読み書き両用として再マウントすることができます。この方法は, 少し極端かもしれませんが, セキュリティ・データを破損から保護する良い方法です。また, リムーバブル・メディアの読み取り専用ロック・タブを物理的に設定することもできます。
- スティッキ・ビット: Tru64 UNIX では, ディレクトリ・エントリにスティッキ・ビットがあります。スティッキ・ビットが設定されていると, ディレクトリ・エントリ (リンク) の削除は, 要求しているユーザがそのディレクトリを所有している場合 (つまり, ディレクトリの所有者) だけに制限されます。この保護がない場合, ディレクトリへの書き込みアクセス権さえあれば, プログラムはディレクトリ・エントリを削除できます。スティッキ・ビットは, 適切に使用してください。たとえば, 1 つのディレクトリに, 異なるユーザが所有するファイルを格納する必要があるプログラムの場合に使用します。

## 1.2 ライブラリとヘッダ・ファイル

システムのドキュメントには、すべての新しいセキュリティ・システム・コール (セクション 2) とルーチン (セクション 3) のリファレンス・ページが含まれています。

ライブラリ `libsecurity.so`, `libaud.a`, `libaud.so`, `libpacl.a`, および `libpacl.so` には、新しく拡張されたセキュリティ・インタフェースのバイナリがあります。これらのライブラリをプログラムにリンクするには、`-l` コンパイル・オプションを使います。たとえば、次のように使用します。

```
$ cc ... -lsecurity -ldb -lm -laud ...
```

プログラムには、いくつかのヘッダ・ファイルを取り込む (インクルードする) 必要があります。このヘッダ・ファイルには、Tru64 UNIX セキュリティ・インタフェースを使うのに必要な定義 (定数、マクロ、構造体、ライブラリ・インタフェースなど) があります。従来の UNIX の慣例に従い、すべての Tru64 UNIX システム・コールとライブラリのリファレンス・ページでは、そのルーチンを使うために必要なヘッダ・ファイルを示しています。多くの場合、次の各ヘッダ・ファイルを、次の順序で使用します。

<code>&lt;sys/secdefines.h&gt;</code>	システムのセキュリティ構成を決定するコンパイル定数を定義します。このファイルは、必ず最初にインクルードする必要があります。
<code>&lt;sys/security.h&gt;</code>	一般的な定義を格納しています。このファイルは、ほとんどの場合にインクルードする必要があります。
<code>&lt;sys/acl.h&gt;</code>	アクセス制御リスト用です。このファイルは、アクセス制御リストを操作する場合に必要です。
<code>&lt;prot.h&gt;</code>	認証データベースと Tru64 UNIX 保護サブシステムを定義しています。プログラムで認証データベースをアクセスする場合に必要です。

<sys/audit.h>	セキュリティ監査インタフェースの監査サブシステム定数を定義しています。監査レコードを生成または処理する場合に必要です。
<protcmd.h>	Tru64 UNIX で提供されるトラステッド・コマンド用のその他の定義がいくつかあります。多くの場合、これらの定義は必要ありません。
<sia.h>	SIA の定数、構造体、およびマクロ定義があります。
<siad.h>	インタフェースおよびセキュリティ・メカニズムで内部的に使用する、SIA の定数、構造体、およびマクロ定義があります。

### 1.3 標準のトラステッド・システム・ディレクトリ

Tru64 UNIX では、セキュリティ情報を格納するためにいくつかのディレクトリが定義されています。これらのファイルとディレクトリの説明は、リファレンス・ページで参照できます (特にリファレンス・ページのセクション 4)。

標準のトラステッド・システム・ディレクトリ内に、新たにファイルおよびディレクトリを作成することが必要な場合もあります。通常、これらのツリーにファイルを置くときは、新しいディレクトリを作成する必要があります。このようなファイル用として作成されているディレクトリの場合以外は、既存のディレクトリに新しいファイルを置かないでください。表 1-1 は、使用する可能性のあるディレクトリを示しています。

表 1-1: 標準のトラステッド・システム・ディレクトリ

ディレクトリ	内容
/tcb/bin , /usr/tcb/bin	直接実行されるトラステッド・コマンドおよびデーモンが置かれます。
/tcb/lib	コマンド行からは起動されず、他のトラステッド・プログラムから実行されるプログラムが置かれます。

表 1-1: 標準のトラステッド・システム・ディレクトリ (続き)

/tcb/files	トラステッド・コンピューティング・ベース (TCB) で使用する制御ファイル、データベース、およびスクリプトが置かれます。必要に応じて、保護サブシステム用にこのディレクトリのサブディレクトリを定義することができます。
/var/tcb	/tcb ディレクトリの代替ディレクトリです。

## 1.4 セキュリティに関連するシステム・コールとライブラリ・ルーチン

以降の項の表では、プログラマから見てセキュリティに関連する Tru64 UNIX システム・コールとライブラリ・ルーチンを示しています。

ただし、これらの項で扱っていないシステム・コールとライブラリ・ルーチンについても、セキュリティに関連するものがあります。

システム・コールやライブラリ・ルーチンを誤って使用し、セキュリティに対する配慮が行われなければ、コンピュータ・システムのセキュリティが脅やかされる可能性があります。たとえば、特権プロセスからシステム・コールが呼び出された場合、ファイル・アクセス許可はチェックされません。最終的には、プログラマがプログラムのセキュリティに関して責任を持つことになります。

### 1.4.1 システム・コール

表 1-2は、プログラマから見てセキュリティに関連するシステム・コールを示しています。

表 1-2: セキュリティに関連するシステム・コール

カテゴリ	システム・コール
ファイル制御	creat , open , fcntl , read , mknod <sup>a</sup> , write
プロセス制御	fork, sigpause , execve , sigsetmask , setpgrp <sup>a</sup> , sigvec , sigblock
ファイル属性	access , chroot <sup>a</sup> , chmod <sup>a</sup> , stat , chown <sup>a</sup> , umask
ユーザ ID とグループ ID	getegid , getuid , getgid, setgroups <sup>a</sup> , geteuid , setreuid <sup>a</sup>

表 1-2: セキュリティに関連するシステム・コール (続き)

カテゴリ	システム・コール
監査	audcntl <sup>a</sup> , audgen <sup>a</sup>
汎用	syscall

<sup>a</sup>これらのシステム・コールを呼び出せるのは、特権プロセスだけです。非特権プロセスから呼び出すと、異なる動作を行う可能性があります。詳細については、関連するリファレンス・ページを参照してください。

1.4.2 ライブラリ・ルーチン

ライブラリ・ルーチンは、プログラムから呼び出せるシステム・サービスです。多くのライブラリ・ルーチンでは、システム・コールを使用します。表 1-3は、セキュリティに関係する Tru64 UNIX ライブラリ・ルーチンを示しています。

表 1-3: セキュリティに関連するライブラリ・ルーチン

カテゴリ	ライブラリ・ルーチン
ファイル制御	fopen , popen
パスワード処理	getpass , putpwent , getpwnam , setpwent , getpwent , endpwent , getpwuid , passlen , pw_mapping , randomword , time_lock
プロセス制御	signal



## トラステッド・プログラミング技法

この章では、トラステッド・プログラムを設計するための技法を紹介しています。この章では、次の内容について説明しています。

- SUID プログラムと SGID プログラムの書き方
- エラー処理
- ファイルの保護
- 安全な探索パスの指定
- シグナルへの応答
- 子プロセスでのオープン・ファイル記述子の使用
- X 環境でのセキュリティ上の考慮事項

### 2.1 SUID プログラムと SGID プログラムの書き方

SUID (セット・ユーザ ID) プログラムと SGID (セット・グループ ID) プログラムでは、プロセスの実効 UID または GID が、プログラムの UID または GID に変更されます。これらのプログラムでは、ファイルの所有者のアクセス権がプロセスに与えられるため、システム・レベルのファイルやディレクトリへのアクセス制御に関する問題のソリューションとなります。

ユーザ ID に `root` が設定されるプログラムや、グループ ID にシステム・レベルのファイルへの書き込みアクセスが可能なグループが設定されるプログラムでは、セキュリティが脅かされる可能性が高くなります。作業を実行する方法が他にない場合以外は、ユーザ ID を `root` に設定するプログラムは作成しないでください。

`chown` システム・コールでは、実行しているプロセスの RUID にゼロが設定されていない限り、ファイルの SUID や SGID ビットが自動的に削除されます。これにより、`root` アカウントが所有する SUID プログラムまたは SGID プログラムを誤って作成することを防止できます。詳細については、`chown(2)` を参照してください。

以下は、さらに安全な SUID プログラムおよび SGID プログラムを作成するためのアドバイスです。

- `access` システム・コールを使って、ユーザによって指定されたすべてのパス名をチェックします。
- コア・ダンプを防ぐために、関連するシグナルすべてをトラップします。
- エラー状態 (たとえば、システム・コールのリターン値やバッファのオーバーフロー) をすべてテストします。

可能であれば、SUID プログラムではなく SGID プログラムとしてください。この理由の 1 つとして、ユーザのファイル・アクセスよりもグループのファイル・アクセスの方が一般的に制限されていることがあげられます。SGID プログラムが傷つけられた場合、ファイル・アクセスが制限されているため、攻撃者が行える動作の範囲が狭くなります。

別の理由として、SGID プログラムを実行したユーザが所有するファイルへのアクセスが比較的簡単であることがあげられます。ユーザが SUID プログラムを実行した場合、元の実効 UID をファイル・アクセスに使用することができなくなります。しかし、ユーザが SGID プログラムを実行した場合、ユーザの 1 次 GID はグループ・アクセス・リストの一部として、利用できる状態のままとなります。このため、SGID プロセスは、ユーザがアクセスできるファイルへのグループ・アクセス権を持ったままとなります。

すべての SUID プログラムのスタックは、省略時は実行可能ではありません。スタックが実行可能であることを前提としているユーザ・アプリケーションは失敗します。どうしても必要な場合は、省略時の設定を変更することができます。これにより、SUID プログラムのスタックを実行可能にすることができます。省略時のゼロ (実行不可) を実行可能に変更するには、次のコマンドを使います。

```
# sysconfig -r proc executable_stack=1
```

リブートしても変更が適用されたままにするには、`sysconfigdb` コマンドを使って、エントリを `/etc/sysconfigtab` ファイルに追加します。

## 2.2 エラー処理

ほとんどのシステム・コールとライブラリ・ルーチンは、呼び出しの成功または失敗を示す、整数のリターン・コードを戻します。必ずリターン・コードをチェックして、ルーチンが正常終了したことを確認してください

い。呼び出しが失敗した場合は、グローバル変数 `errno` を調べて、失敗の原因を見つけてください。

`errno` 変数が設定されるのは、システム・コールでエラーが発生したときです。この値を使用すると、エラー状態の詳細を知ることができます。この情報は、プログラムでの対応方法を決定したり、より詳しい診断メッセージを作成するのに役立ちます。このエラー・コードは、`<errno.h>` 内のエラー名に対応しています。詳細については、`errno(2)` を参照してください。

次の `errno` 値は、セキュリティ侵害の可能性を示します。

EPERM	ファイルの所有者またはスーパーユーザだけが許されている方法で、所有者以外のユーザがファイルを変更しようとしたことを示します。また、スーパーユーザだけが許されている方法で、ユーザが何らかの操作を行おうとしていたことを示す場合もあります。
EACCES	許可されていないファイルをユーザがアクセスしようとしたことを示します。
EROFS	マウントされたファイル・システムのアクセス許可が無効になっているときに、そのファイル・システム上のファイルにアクセスしようとしたことを示します。

特権システム・コールを使用するプログラムは、生成された実行可能プログラムにスーパーユーザ特権がない場合、特権システム・コールを実行しようすると失敗します。セキュリティ管理者が特権システム・コールの実行失敗を記録するように監査システムを設定している場合は、失敗が記録されます。

プログラムがセキュリティの侵害と考えられる現象を検出した場合、攻撃者がこのプログラムを攻撃するために役立つような診断メッセージを表示しないでください。たとえば、攻撃者の実ユーザ ID (UID) がアクセス対象ファイルの UID と一致しなかったためにプログラムが終了するというメッセージを表示したり、アクセス対象ファイルの名前を表示したりしないでください。この情報は、SUID root プログラムの場合は `audgen()` ルーチンを使い、その他のプログラムの場合は `syslog` を使うことで制限してください。その他、メッセージを表示する前に遅延を少し入れることで、さまざまな入力データを系統的に試してプログラムに侵入する、プログラム化された侵入操作を防ぐことができます。

## 2.3 ファイルの保護

プログラムが永続ファイル (たとえば、データベース) を使用する場合は、これらのファイルのアクセス許可を制限し、プログラムでアクセスを制御するようにしてください。これらの注意事項は、共用メモリ・セグメント、セマフォ、およびプロセス間通信のメカニズムに対しても適用されます。これらのすべてのオブジェクトには、制限された許可を設定してください。

プログラムは、実行中にデータを格納するため、一時ファイルを作成することがあります。一時ファイルを使う場合は、次の注意事項に従ってください。

- プログラムが終了する前に、一時ファイルを削除するようにします。
- 情報が暗号化されていない場合、一時ファイルへの機密情報の格納は避けます。
- 一時ファイルの所有者だけに、読み取りおよび書き込みの許可を与えます。プログラムの先頭で `umask()` システム・コールを使用して、ファイル作成マスクに `077` を設定します。
- 一時ファイルは、所有者だけが書き込めるプライベート・ディレクトリまたは `/tmp` に作成します。`/tmp` ディレクトリには、スティッキ・ビットが設定されています (モード `1777`)。このため、このディレクトリ内のファイルは、ファイルの所有者、ディレクトリの所有者、またはスーパーユーザだけが削除できます。

一般的な方法は、一時ファイルを作成し、そのファイルがまだオープンされている状態でリンクを解除 (`unlink`) することです。これにより、アクセスは、リンクを解除する前にファイルをオープンしたプロセスだけに限定されます。プロセスの終了時には、`i` ノードが解放されます。

NFS マウントされたファイル・システム上で `unlink` をこのように使用すると、少し異なる動作になります。クライアントのカーネルはファイルの名前を変更し、プロセスの終了時に NFS にのみリンク解除が送られます。他者からそのファイルにアクセスできなくすることはできません。ただし、プロセスが終了したときには、そのファイルは確実にアクセスできなくなります。いずれにしても必ず明示的に、プロセス終了時後には一時ファイルが残らないようにしてください。

## 2.4 安全な探索パスの指定

`popen` , `system` , または `exec*p` ルーチンを使用して `/bin/sh` または `/sbin/sh` を実行する場合は、パス名の指定やシェル変数 `PATH` の定義を慎重に行ってください。 `PATH` 変数は、システム上のコマンドとスクリプトを実行するための探索パスを指定するため、セキュリティ上重要な変数です。詳細については、`environ(7)` , `popen(3)` , および `system(3)` を参照してください。

安全な探索パスを作成する方法を、次に示します。

- `PATH` 変数に、絶対パス名を指定します。
- 公用ディレクトリ、一時ディレクトリ、他のユーザのディレクトリ、または現在の作業ディレクトリを探索パスに含めないでください。これらのディレクトリを含めると、誤ったプログラムを不注意で実行したり、不正なプログラムによってトラップされる可能性が高まります。
- システムのディレクトリを、ユーザのディレクトリよりも前にリストします。これにより、システム・プログラムと同じ名前のプログラムをユーザが誤って実行するのを防ぎます。
- パス・リストの構文 (特に空、小数点、およびコロンの使い方) を解析してください。パス・リスト内の空エントリまたは小数点エントリは、現在の作業ディレクトリを指定します。またコロンは、パス・リスト内のエントリを区切るために使います。このため、等号に続く最初のエントリをコロンで始めてはなりません。
- パス・リストがコロンで終わっている場合、一部のシェルと `exec*p` ルーチンは、現在の作業ディレクトリを最後に探索します。各種シェルがこの最後のコロンを異なる方法で解釈するのを避けるには、空エントリの代わりに小数点を使って現在の作業ディレクトリを参照します。

`execve` システム・コールにはパス名を指定する必要があるため、`exec*p` ルーチンではなく、`execve` を使用の方が良い場合もあります。詳細については、`execve(2)` を参照してください。

## 2.5 シグナルへの応答

Tru64 UNIX オペレーティング・システムは、特定のイベントに対する応答としてシグナルを生成します。イベントは、ユーザの端末 (たとえば、終了、割り込み、または停止)、プログラムのエラー (たとえば、バス・エラー)、または別のプログラム (たとえば、kill) により起動されます。

特に指定しなければ、ほとんどのシグナルで受信プロセスが終了します。しかし、いくつかのシグナルでは受信プロセスは停止するだけです。SIGQUIT や SIGTRAP などの多数のシグナルでは、デバッグの目的でコア・イメージがファイルへ書き込まれます。コア・イメージ・ファイルには、パスワードのような機密情報が含まれている可能性があります。

コア・イメージ・ファイル内の機密情報を保護し、キーボードからの入力による割り込みからプログラムを保護するには、SIGQUIT、SIGTRAP、または SIGTSTP などのシグナルを捕捉するプログラムを書きます。

signal ルーチンを使って、シグナルに対するプロセスの応答を変えるようにします。これらのルーチンによってプロセスは、シグナルが送信されたときにそれを無視したり、サブルーチンを呼び出したりといったことが可能です。(SIGKILL シグナルおよび SIGSTOP シグナルのキャッチ、無視、ブロックはできません。これらのシグナルは必ず、受信プロセスに渡されます)。詳細については、signal(3) および sigvec(2) を参照してください。

子プロセスは、親プロセスが fork を呼び出す前に設定したシグナル・マスクを継承します。execve システム・コールは、捕捉済みのすべてのシグナルを省略時の動作にリセットします。無視されるシグナルはそのまま無視されます。このため、fork または execve を呼び出す前に、プロセスが適切にシグナルを処理するようにしてください。詳細については、fork(2) および execve(2) のリファレンス・ページを参照してください。

## 2.6 子プロセスでのオープン・ファイル記述子の使用

子プロセスは、その親プロセスのオープン・ファイル記述子すべてを継承することができます。このため、同じようにファイルにアクセスすることができます。この関係により、セキュリティ上の考慮が必要となります。

たとえば、次の処理を行う、セット・ユーザ ID (SUID) プログラムを作成するとします。

- 機密性の高い特権ファイルへのデータの書き込みをユーザに許可する。
- 非特権状態で実行される子プロセスを作成する。

親の SUID プロセスは書き込み用にファイルをオープンするため、子プロセス(または、子プロセスを実行しているすべてのユーザ)は、重要なファイルに書き込みを行うことができます。

機密性の高い特権ファイルの子プロセスのユーザから守るには、子プロセスを作成する前に、子プロセスに不要なファイル記述子をすべてクローズします。子プロセスを作成する前にファイル記述子をクローズする効果的な方法は、`fcntl` システム・コールを使うことです。この呼び出しを使用すると、ファイルをオープンした後で、そのファイルに `close-on-exec` フラグを設定することができます。このフラグが設定されているファイル記述子は、`exec` システム・コールで新たなプログラムをプロセスが開始するときに、自動的にクローズされます。

詳細については、リファレンス・ページの `fcntl(2)` を参照してください。

## 2.7 X 環境でのセキュリティ上の考慮事項

以降の項では、X プログラミング環境でセキュリティを強化する方法について説明しています。

- アクセス制御の制限
- キーボード入力のプロtection
- キーボードおよびマウスのイベントのブロック
- デバイス関連イベントの保護

### 2.7.1 キーボード入力のプロtection

アクセス制御リストにリストされたホストへログインしたユーザは、`XGrabKeyboard` 関数を呼び出してキーボードの制御を行うことができます。クライアントがこの関数を呼び出すと、X サーバはすべてのキーボード・イベントをそのクライアントだけに送ります。攻撃者はこの呼び出しを使用して、あるウィンドウからの入力ストリームを取得し、別のウィンドウへ転送することができます。攻撃者は、シミュレートされたキーストロークをウィンドウへ戻し、このウィンドウを実行しているユーザをだますことができます。このためユーザは、不正が行われていることに気付かない可能性があります。

攻撃者がユーザのキーストロークを取り込むことができると、ワークステーション上に格納されたデータの機密性が脅かされます。

X ウィンドウには、セキュア・キーボード・モードがあります。このモードでは、ワークステーションのキーボードからのユーザの入力がすべて、1 つの安

全なウィンドウへ送られます。ユーザがこのモードを設定するには、X ウィンドウの [Command] メニューから [Secure Keyboard] 項目を選択します。

機密データを扱うプログラムには、セキュア・キーボード・モードを含めてください。この予防策は、プログラムがユーザにパスワードの入力を求める場合に特に重要です。

セキュア・キーボード・モードを実現するためのガイドラインを、次に示します。

- Xlib の XGrabKeyboard 呼び出しを使います。
- セキュア・キーボード・モードが設定されていることを、ビジュアルな方法でユーザに知らせます。たとえば、画面上で反転表示を使います。
- ユーザがウィンドウをアイコン化したときに、XUngrabKeyboard 関数を使ってキーボード・グラブを解放します。キーボードが解放されると、キーストロークを別のウィンドウへ転送できるようになります。

## 2.7.2 キーボードおよびマウスのイベントのブロック

アクセス制御リストにリストされたホストは、ウィンドウの ID を知っていれば、どのウィンドウにもイベントを送ることができます。XSendEvent 呼び出しを使用すると、呼び出し側アプリケーションは、キーボードまたはマウスのイベントを指定のウィンドウへ送ることができます。攻撃者はこの呼び出しを使用して、破壊を行うためのデータをウィンドウへ送ることができます。たとえばこのデータにより、`rm -rf *` コマンドを実行したり、機密ファイルの内容をテキスト・エディタを使って変更したりすることがあります。端末がアイドル状態の場合、ユーザはこれらのコマンドが実行されていることに気付かない可能性があります。

攻撃者が破壊を行うためのデータをワークステーションのウィンドウへ送ることができると、ワークステーション上に格納されたデータの完全性が脅かされます。

.Xdefaults ファイル内の `allowSendEvents` リソースに `False` が設定されている場合、X ウィンドウは、別のクライアントから送られて来たキーボードおよびマウスのイベントをブロックします。

別のクライアントから送られて来たイベントをブロックするプログラムを書くこともできます。XSendEvent 呼び出しは、指定されたウィンドウへイベントを送り、イベント構造体の `send_event` フラグに `True` を設定します。プ



ログラムでは、受け付けた各キーボードおよびマウスのイベントについて、このフラグをテストします。このフラグに `False` が設定されている場合、イベントはキーボードで生成されたものなので、受け付けても安全です。

### 2.7.3 デバイス関連イベントの保護

キーボードおよびマウスのイベントのようなデバイス関連イベントは、次の条件のいずれかを満たすまで、ソース・ウィンドウから祖先のウィンドウへと上方へ伝えられます。

- Xクライアントがイベント・マスクを設定することにより、ウィンドウに対してそのイベントを選択する。
- Xクライアントがそのイベントを `do-not-propagate` マスクに含めることにより、そのイベントを拒否する。

`XReparentWindow` 関数を使って、ウィンドウの親を変更することができます。この呼び出しは、ウィンドウの親を、同じ画面上の別のウィンドウに変更します。ウィンドウの親を変更する上で知っておく必要があるのは、ウィンドウ ID だけです。子のウィンドウ ID を使って、その親のウィンドウ ID を見つけることができます。

`XReparentWindow` の呼び出しを誤って使用すると、ウィンドウ・システムのセキュリティが脅かされます。新しい親ウィンドウは、子ウィンドウが選択しなかったイベントを選択することができます。

この種の悪用から守るためには、次のような予防策をとってください。

- 子ウィンドウに、必要なデバイス・イベントを選択させます。この予防策は、子から上方へ伝達されたイベントが新しい親に傍受されるのを防ぎます。子ウィンドウのイベント処理を親ウィンドウに集中化する方法は、セキュリティ上大きな危険が伴います。攻撃者は、親ウィンドウを変更して、子ウィンドウに対するイベントを傍受することができます。このため、各子ウィンドウ自身で自分のデバイス・イベントを処理する方が安全です。子ウィンドウが明示的に選択したイベントは、伝達されません。
- デバイス・イベントがウィンドウ階層内でそれ以上伝達されないよう、子ウィンドウに指定させます。指定するには、`do-not-propagate` マスクを設定します。この予防策により、子ウィンドウがイベントを必要

とするかどうかに関係なく、デバイス・イベントが親ウィンドウへ伝達されるのを防ぐことができます。

- 親ウィンドウが変更されたときに、子ウィンドウに通知されるようにします。通知を受けるには、子ウィンドウのイベント・マスクの `StructureNotify` または `SubstructureNotify` ビットを設定します。これらのイベント・マスクの設定についての詳細は、『*X Window System: The Complete Reference to Xlib, X Protocol, ICCCM, XLFd*』を参照してください。

## 2.8 シェル・スクリプトの保護

機密データを処理するシェル・スクリプトを書くときは、スクリプトの本体より前で `PATH` 変数を設定してエクスポートします。シェル・スクリプトには `SUID` や `SGID` は設定しないでください。

---

## 認証データベース

この章では、次の内容について説明しています。

- 認証データベースの概要
- 認証データベースの構成要素
- 認証データベースのアクセス

### 3.1 認証データベースの概要

認証データベースは、エンハnst・セキュリティが使用可能になっているときに、Tru64 UNIX のセキュリティ情報をすべて格納する一連のデータベースです。認証データベースは、次のデータベースから構成されています。

- デバイス割り当て
- ファイル制御
- システム省略時設定
- 保護パスワード
- 端末制御

エンハnst・セキュリティが使用可能なシステム用に特別に作成されたトラステッド・プログラム (つまり、セキュリティ・ルールを破ることができる立場にあるプログラム) は、これらのデータベース内の情報を使用する必要があります。特別な場合を除き、システム管理者は Tru64 UNIX 管理インタフェースを使ってこれらのデータベースを保守します。このため、ユーザのプログラムでは通常、これらのデータベースの読み取りのみを行います。

以下の節では、認証データベースについて説明します。ファイル形式についての一般的な説明は、authcap(4) のリファレンス・ページを参照してください。

### 3.1.1 デバイス割り当てデータベース (devassign)

デバイス割り当てデータベースには、システム上のデバイスのデバイス属性が格納されています。現在のところ、devassign データベースに入っているデバイスは、次の 2 種類です。

- 端末
- X ディスプレイ

デバイス関連コマンドには、デバイス・エントリの名前を使用します。この名前は、デバイスを表すデバイス・ファイルとは関係ありません。

システム管理者は、デバイス割り当てデータベースを保守します。ユーザのプログラムでは、このデータベースの内容を変更してはなりません。

このデータベースの論理エントリのサイズは、動的に変わります (自己内蔵型ではありません)。このため、そのエントリを含む構造体の作業用コピーを作るには、`copiesdvent()` ルーチンを使わなければなりません。詳細については、`getesdvent(3)` のリファレンス・ページを参照してください。

テキスト・ファイル `/etc/auth/system/devassign` には、デバイス割り当てデータベース全体が格納されています。

### 3.1.2 ファイル制御データベース

ファイル制御データベースは、機密ファイルの保護属性 (所有者、モード・ビットなど) が正しいか確認するのに役立ちます。このデータベースには、各ファイル (またはディレクトリ) の絶対パス名と正しい属性が保存されています。これらの属性には、次の項目の任意の組合せが入っています。

- ファイル・タイプ (通常、ブロック型特殊、文字型特殊、ディレクトリ、FIFO、ソケット)
- 所有者
- グループ
- 許可モード・ビット
- アクセス制御リスト (システムにアクセス制御リストが構成されている場合)

ユーザのプログラムでは、`create_file_securely()` インタフェースを介して新しく作成されたファイルに対してファイル制御データベースのエント

リを使用する以外は、ファイル制御データベースからの読み取りまたはファイル制御データベースへの書き込みを行ってはなりません。ただし、すべての新しい機密ファイルと機密ディレクトリをデータベースに追加する必要があります。変化しない属性をすべて含めます。これにより、これらの属性が定期的にチェックされ修正されるようになります。

`create_file_securely()` ルーチンを使って、ファイル制御データベース内に指定された属性でファイルを作成することができます。このルーチンを使用できるのは、新しいファイルを作成する場合だけです。新しいバージョンのファイルを、別のファイルに作成する必要があります。(Tru64 UNIX の表記法により、ファイルの新しい内容を表す `:t` をパス名に追加します)。次に、新しいファイルの名前を既存のファイル名に変更します (`rename()` システム・コールを使用)。

ファイル制御データベースは、テキスト・ファイル `/etc/auth/system/files` です。このファイルのフォーマットの定義は、`files(4)` のリファレンス・ページを参照してください。システム管理者は、`edauth -df` コマンドを使って、データベースへエントリを追加したり、データベースからエントリを削除することができます。詳細については、`edauth(8)` のリファレンス・ページを参照してください。

### 3.1.3 システム省略時設定データベース (default)

システム省略時設定データベース `/etc/auth/system/default` は、対応するフィールドが他のデータベースで未定義のままのときに使用されるフィールドが格納されたテキスト・ファイルです。つまり、このデータベースには、エンハンスト (保護) パスワード、デバイス割り当て、および端末制御データベースの省略時の情報が入っています。(各認証データベース内のフィールドはどれも未定義にできますが、すべてのフィールドに省略時の値があるわけではありません)。

システム省略時設定データベースには、システムのその他のパラメータのフィールドもあります。ユーザのプログラムには、その他の情報は必要ありません。

システム管理者がこのデータベースを保守するため、ユーザのプログラムでこのデータベースを変更することはありません。他のデータベースのアクセス・ルーチンも、システム省略時設定を戻します。システム省略時設

定データベースの情報にアクセスし、使用方法の例は、3.1.5 項を参照してください。

システム省略時設定データベース全体には、エントリが 1 つだけあります (default エントリ)。

### 3.1.4 エンハンスト (保護) パスワード・データベース

エンハンスト・パスワード・データベース (/tcb/files/auth.db および /var/tcb/files/auth.db) は、一連のユーザ認証プロファイルが格納された dbm ファイルです。ユーザ認証プロファイルは、NIS prpasswd マップを使って、Tru64 UNIX システム間で配布することもできます。各認証プロファイル・エントリには、ユーザ名 (ログイン時にユーザが使う名前) で名前が付けられています。認証プロファイルには、ユーザのログイン・セッションを管理するフィールドが多数あります。これらのフィールドについての詳細は、第 4 章を参照してください。

認証プロファイルは、従来の /etc/passwd ファイルまたは NIS passwd マップの 1 行で存在が示されるアカウントに対応しています。暗号化されたパスワードは、/etc/passwd ファイルから認証プロファイルに移動されました。

システムは、/etc/passwd データベース内のその他のフィールドを、従来通りの意味で扱います。/etc/passwd 内の各エントリは、保護パスワード・データベース内でユーザ ID と名前が同じ認証プロファイルと正確に 1 対 1 で対応しています (有効なアカウントとしてみなすには、両方が必要)。/etc/passwd エントリには、ダミーの暗号化パスワード・フィールドがあります。認証プロファイルには、本当のパスワード・フィールドがあります。

/etc/passwd ファイルを照会する従来の UNIX インタフェースは、getpwent() です。インタフェースの機能に違いはなく、情報は必ず /etc/passwd ファイルまたは NIS マップから取り出されます。ただし、戻される暗号化パスワードはダミーの値であることに注意してください (このルーチンに対して、認証プロファイルから暗号化パスワードを取り出すような変更は行われていません)。

ユーザのプログラムでは、エンハンスト (保護) パスワード・データベースを変更してはなりません。ただし、多くのトラステッド・プログラムでは、ユーザ認証プロファイルから情報を読み取る必要があります。

### 3.1.5 端末制御データベース

端末制御データベース (/etc/auth/system/ttys.db) は、主にログイン時に使用され、ログインしているユーザに対してではなく、ログイン端末に対して適用されるフィールドが格納された dbm ファイルです。このデータベースは、ユーザがログインできる各端末 (X 端末を含む) のエントリで構成されています。

データベース内の各エントリには、端末の名前があります。この名前は、ログイン・ポートを指定するために使用されるファイル (/etc/inittab) 内の名前に対応しています。デバイス割り当てデータベースのエントリは、端末制御データベースの各エントリと対応します。対応するエントリがデバイス割り当てデータベースにない場合、ほとんどのトラステッド・プログラム (たとえば、login) はサービスを提供しません。

各端末制御データベースのエントリには、次のフィールドがあります。

- 端末の名前。
- ユーザ ID と、最後にログインに失敗した日時。ユーザ ID はデータベースに格納されているため、ユーザ ID に対応しないユーザ名を指定したログイン失敗では、このデータベースには有効なユーザ ID は設定されません。有効な ID にユーザ名が対応している場合、その ID は適切なフィールドに置かれます。
- ユーザ ID と、最後に正常にログインした日時。
- 最後にログインが成功してから、ログインが失敗に終わった回数。
- 端末がロックされているかどうか。
- システムで許されているログイン失敗の回数。この回数を超えると、端末がロックされます。
- ログインに失敗した後の強制遅延 (ログイン・プログラムによる強制)。
- ログインの開始後、キーボード入力がない場合にログインがタイムアウトするまでの時間 (秒数)。タイムアウトになると、ログイン・プログラムはログイン処理を終了します。

Tru64 UNIX のログイン・プログラムでは多数のフィールドを変更しますが、このデータベースのエントリを保守するのはシステム管理者です。ユーザのプログラムは通常、このデータベースを変更しません。あまりないこと

ですが、トラステッド・プログラムでは、このデータベースを読み取らなくてはならないこともあります。

/etc/auth/system/ttys.db ファイルには、端末制御データベース全体が格納されています。

## 3.2 認証データベースの構成要素

各データベースは、名前付きのエントリ群で構成されています。プログラムでは、データベース内のエントリを順次検索することもできますが、主にエントリの名前を使ってデータベースから特定のエントリを取り出します。

各エントリには、一連のフィールドが含まれています。各フィールドには、フィールドと値にアクセスするための識別子があります。各データベースでは、そのエントリに使用可能なフィールドが決まっています。個々のフィールドは省略可能で、エントリから省略することができます。フィールドの種類には、文字列、整数、論理値があります。

エントリの一般的な形式を次に示します。

```
entry_name:string_field=value:integer_field#value:\
:boolean_field_true:boolean_field_false@:chkent:
```

ライブラリ・ルーチンは通常、1つのエントリ全体を読み取ったり書き込んだりします。Cの構造体には、指定されたデータベース・エントリが持つ可能性のあるフィールドすべてが確保されています。この構造体には必ず、フラグ構造体が付いています。このフラグ構造体は、読み取ったり書き込んだりするフィールドを示すマスクを持っています。

フィールドが未定義の場合、ユーザのプログラムは適切な動作を行う必要があります。多くの場合、未定義フィールドはシステム省略時設定データベースから取り出されます。これについては 3.2.1 項 で説明します。各データベースの構造体には、そのデータベース用のシステム省略時設定フィールドとフラグがあります。このため、システム省略時設定は個々のエントリの値が格納されている構造体と同じ構造体から利用でき、特定のフィールドに関連するシステム省略時設定の値を簡単に取り出すことができます。

### 3.2.1 データベースの形式

通常は、認証データベースの物理的フォーマットを意識する必要はありません。すべてのデータベースの論理的フォーマットは同じで、アクセス・ライブラリも似ています。たとえば、端末制御データベースは、各制御対象端末



用のエントリで構成されています。次の ttys ファイルの例は、tty01 の物理的フォーマットのエントリを示しています。また、次の表は、データベース・ファイルのフォーマットを示しています。

```
tty01:t_devname=tty01:t_uid#44:t_logtime#772479074:\
:t_login_timeout#20:t_failures#3:t_lock@:\
:chkent:
```

意味	フィールド	値	説明
名前	t_devname	tty01	端末 1
最後にログインしたユーザ	t_uid	44	UID が 44
最後にログインした日時	t_logtime	772479074	Fri Jun 24 13:31:13 EDT 1994
ログイン・タイムアウト	t_login_timeout#20	20	ログイン・タイムアウト (秒)
最後にログインした後の失敗回数	t_failures#3	3	最後に正常にログインした後にログインに失敗した回数
アカウントの状態	t_lock	@	アンロック (偽)
チェック・エントリ	:chkent:<EOL>	chkent	エントリの最後

次の C の構造体は、ttys データベースからエントリを取り出すために使用します (インクルード・ファイル <prot.h> を参照)。

```
struct es_term {
    struct estc_field *ufld;    /* fields for this entry */
    struct estc_flag *uflg;    /* flags for this entry */
    struct estc_field *sfld;    /* system default fields */
    struct estc_flag *sflg;    /* system default flags */
};
```

estc\_field は、エントリのフィールドのデータを保持しています。  
estc\_flag は、estc\_field 内に存在する (つまり設定されている) フィールドを示すフラグを保持しています。次に、estc\_field 構造体を示します。

```
struct estc_field {
    char *fd_devname;    /* terminal name */
    uid_t fd_uid;    /* uid of last successful login */
    time_t fd_slogin;    /* time of last successful login */
    ushort fd_nlogins;    /* consecutive failed attempts */
    char fd_lock;    /* terminal lock status */
};
```

```

        ushort   fd_login_timeout; /* login timeout value */
    };

    struct estc_flag {
    unsigned short
        fg_devname      :1,      /* name present? */
    /* fg_uid           :1,      /* uid present? */
    /* fg_slogin        :1,      /* time present? */
    /* fg_nlogins        :1,      /* failed attempts present? */
    /* fg_lock           :1,      /* lock status present? */
    /* fg_login_timeout :1       /* login timeout present? */

    };

```

gettestcent(3) のリファレンス・ページでは、端末制御データベースにアクセスするために使用できるライブラリ・ルーチンが定義されています。このアクセス・ルーチンは、特定のエントリ `ufld` および `uflg` のフィールドや、システム省略時設定 (`sfld` および `sflg`) のフィールドを取り出したり、設定したりします。システム省略時設定があるフィールドを持つデータベースの場合、エントリに対応するフィールドの他、システム省略時設定も戻されます。

### 3.2.2 データベースの読み取りと書き込み

各データベースは、ユーザおよびグループが所有しています。ユーザのプログラムは、このユーザおよびグループへ自由にアクセスできなければなりません。プログラムは、2つの方法でインストールすることができます。

- サブシステムの標準プログラムとして、適切なグループへの SGID を設定する。
- サブシステムの標準プログラムとして、SUID 0 を設定する。

ライブラリ・ルーチンは、1つのデータベースへの書き込みプロセスを、一時点では1つに自動的に制限します。ただし、データベースがロックされるのは、書き換えが行われている間だけです。取り出し操作と書き込みの操作にまたがってエントリをロックする方法はありません。書き込みは、できるだけ早く完了させる必要があります。

#### 3.2.2.1 バッファ管理

データベース・エントリの取り出し、置き換え、追加を行うプログラムを適切にコーディングするには、システムがどのようにデータベース・エントリのバッファを割り当てて返すかを知っておかなければなりません。すべて

のデータベース・ルーチンは、`getpwent()` ルーチンを見本にして作成されています。このルーチンは、各呼び出しで再使用される静的記憶域へのポインタを戻します。別のエントリを取り出してから前のエントリを再度参照する必要がある場合、または既存エントリを再度書き込むか新しいエントリを追加する必要がある場合は、バッファの内容を保存しなければなりません。データベース・エントリを読み取り、1 つ以上のフィールドとフラグの値を変更して、データベースを変更するルーチンへ同じバッファを渡すことはできません。

いくつかのデータベース・フィールドの論理的な形式は、自己内蔵型です。その他のフィールドには、可変長データへのポインタが入っています。

`devassign` データベースの論理的な形式には、可変長データへのポインタであるフィールドがあります。`getesdvent(3)` のリファレンス・ページでは、`copiesdvent()` ルーチンについて説明しています。このルーチンは、デバイス割り当てのデータベース・エントリを持つ構造体を割り当て、その中に、`getesdvent()` または `getesdvnrm()` から戻されたバッファの内容をコピーします。

自己内蔵型データベースのエントリは、次のように、構造体の簡単な代入で保存することができます。

```
struct es_passwd *pr;          /* returned value */
struct es_passwd *pwcop;       /* buffer for saved values */

/* Retrieve john's protected password database entry */

pr = getesnam("john");

/* store values of john's entry to a local buffer */

pwcop = copiespwent(pr);
if (!pwcop) abort();

/* Change the password minimum change time to two weeks */

pwcop->uflg->fg_min = 1;
pwcop->uflg->fd_min = 14 * 24 * 60 * 60;

/* Rewrite john's protected password database entry */

if (!putespwnam("john", pwcop))
    errmsg("Could not write protected password entry\n");
free(pwcop);
```

### 3.2.2.2 名前または ID でのエントリの読み取り

データベース・エントリを読み取るには、エントリの名前を指定します。一部のデータベースでは、名前以外の識別値を指定して読み取ることもできます。たとえば、エンハンスト (保護) パスワード・データベースからエントリ名 (ユーザの名前) またはユーザ ID でエントリを取り出すことができます。次のコードは、端末制御データベースから、名前が `tty44` のエントリを読み取ります。

```
.
.
.
struct es_term    *entry;
.
.
.
if ((entry = getestcnam("tty44")) == NULL)
    errmsg ("Entry not found");
```

`getestcnam()` は、戻されるエントリ用のデータ構造体を割り当てます。つまり、`entry` は、`es_term` 構造体へのポインタです。この構造体は、次回、`prtc()` または `estc()` ルーチンが呼び出されたときに再度使用されます。

### 3.2.2.3 エントリの順次読み取り

次に示すコードのようにして、データベース・エントリを順次読み取ることもできます。

```
.
.
.
struct es_term    *entry;
.
.
.
setprtcnt();                                /* rewind the database*/
while ((entry = getestcent()) != NULL) {    /* read next entry   */
.                                           /* process the entry */
.
.
}
endprtcnt ();                                /* close */
```

`getestcent()` もまた、エントリのデータ構造体を割り当てることに注意してください。`setprtcnt()` を使用すると、先頭から検索を再度開始することができます。

#### 3.2.2.4 システム省略時設定の使用

システム省略時設定のフィールドは、エントリ内の対応するフィールドが定義されていないときに使用されます。システム省略時設定データベースには、他のデータベースの省略時の値が入っています。次のデータベースには、システム省略時設定の情報が入っています。

- 保護パスワード
- 端末制御
- デバイス割り当て

省略時設定があるのは、これらのデータベース内の一部のフィールドだけです。

プログラムが論理エントリを読み取った場合、ライブラリ・ルーチンは、そのエントリのフィールド (ufld および uflg) とシステム省略時設定 (sfld および sflg) の両方を戻します。要求したフィールドがエントリにない場合、システム省略時設定を使用します。システム省略時設定も未定義の場合は、プログラムは監査データを生成し、エラーを報告して障害処理を実行する必要があります。それ以外の場合は、省略時の値を安全に定義することができます。

たとえば、端末 `tty14` のタイムアウト値を調べる必要がある場合、コードは次のようになります。

```
struct es_term  *entry;          /* the entry for the terminal */
ushort          time_out;        /* final timeout value */
.
.
.

/*--- fetch the entry by name ---*/

if ((entry = getestcnam ("tty14")) == NULL)
    errmsg ("Entry not found");

/*--- if defined for the terminal, use it ---*/

if (entry->uflg->fg_login_timeout)
    time_out = entry->ufld->fd_login_timeout;

/*--- else if system default is defined, use it ---*/

else if (entry->sflg->fg_login_timeout)
    time_out = entry->sfld->fd_login_timeout;
```

```

/*--- otherwise, assume a value of 0 ---*/

else time_out = 0;

```

### 3.2.2.5 エントリの書き込み

ユーザのプログラムでデータベースを変更することは、めったにありません。またそれ以上に、ユーザのプログラムでシステム省略時設定を変更することはまずありません。ただし、この変更が必要な場合は、新しいフィールドを `ufld` に置き、対応するフラグを `uflg` に設定してから、適切なライブラリ・ルーチン呼び出します。たとえば、端末 `tty14` の新しいタイムアウト値に 20 を設定するコードは、次のようになります。

```

struct es_term  *entry, *ecopy;
.
.
.

/*--- fetch the entry by name ---*/

if ((entry = getestcnam("tty14")) == NULL)
    errmsg ("Entry not found");

/*--- change the desired field(s) ---*/

ecopy = copyestcent(entry); if (!ecopy) abort():
ecopy->ufld->fd_login_timeout = 20; /* set timeout value */
ecopy->uflg->fg_login_timeout = 1; /* set flag to show the
                                   field has been set */

/*--- update the database ---*/

if (!putestcnam("tty14", ecopy))
    errmsg ("Could not update database");
free(ecopy);

```

---

#### 注意

---

後で使用するためにデータを保存するには、適切な `copies*()` ルーチン呼び出さなければなりません。

`copies*()` ルーチンは、`malloc()` 記憶域へのポインタを戻します。この記憶域は、呼び出し側が解放しなければなりません。

---

システムの省略時の値を設定できるのは、システム省略時設定データベース用の `putesdfnam()` インタフェースを使用する方法だけです。たとえば、`es_term` エントリ内のフィールド `sfld` と `sflg` を設定してから、`putestcnam()` を呼び出してシステム省略時設定値を設定することはできません。

### 3.3 認証データベースのアクセス

Tru64 UNIX には、各データベースへアクセスするライブラリ・ルーチンが一式あります。次に示すリファレンス・ページでは、これらのデータベースの形式と使用方法について説明しています。この章とこれらのリファレンス・ページを合わせて参照してください。

名前	データベース	リファレンス・ページ
デバイス割り当て	<code>devassign</code>	<code>getesdvent(3)</code>
ファイル制御	<code>file</code>	<code>getesfient(3)</code>
システム省略時設定	<code>default</code>	<code>getesdfent(3)</code>
保護パスワード	<code>auth.db/prpasswd (NIS)</code>	<code>getespwent(3)</code>
端末制御	<code>ttys.db</code>	<code>getestcent(3)</code>

これらのリファレンス・ページで定義されているライブラリ・ルーチンは、データベースの実際のファイル形式を隠しています。トラステッド・プログラムでは、この形式を意識する必要はありません。これらのライブラリ・ルーチンを使用するだけで済みます。





この章では、次の内容について説明しています。

- 監査 ID
- 識別サポート・ライブラリ
- デーモンの使用
- エンハンスト (保護) パスワード・データベースの使用

## 4.1 監査 ID

Tru64 UNIX では、従来のプロセスのユーザ ID およびグループ ID をすべて維持しています。この他に、Tru64 UNIX 独自の、プロセスごとの監査 ID (AUID) をサポートしています。AUID は基本的には、実ユーザ ID と似ています。ただし、実ユーザ ID が変更される場合でも変更されないところ異なります。

すべての監査レコードには監査 ID が関連付けられています。この監査 ID があることで、実ユーザ ID および実効ユーザ ID がログイン時の値から変更されていても、ユーザを識別できます。

監査 ID は、プロセスの特権に関係なく、プロセスの子孫全体にわたって、1 度だけ設定できます。ログイン時、監査 ID には認証ユーザ (実ユーザ ID および実効ユーザ ID と同じ) が設定されます。プロセスが `fork()` システム・コールを呼び出して子プロセスを生成すると、この監査 ID は親から子へ引き継がれます。

スタートアップ・スクリプトから生成されるプログラムや、`inittab` ファイルの `respawn` エントリによって生成されるプログラムは、監査 ID が未設定の状態で作成されます。(このようなプログラムは通常、インタフェースを通して認証されたユーザに対して AUID を設定する認証プログラム (`getty/login` シーケンス、ウィンドウ・マネージャ、トラステッド・パス・マネージャ) です。

スタートアップ・スクリプトを通して開始されるプログラムは通常、ユーザのためにサービス要求を受け取り、その要求のサービスを行うプロセスを生成します。このようなプログラムは通常、要求側のプロセスの実効 ID をベースとして、子サービス・プロセスの監査 ID を設定します。この種のプログラムを書く場合は、SIA ルーチンを使用する必要があります。SIA ルーチンは、システム上で使用されているセキュリティ・メカニズム (BASE, 拡張, DCE など) に関係なく、子プロセス内のユーザ環境を正しく設定します。

システム・コール `getluid()` および `setluid()` は、監査 ID を読み取り、設定します。詳細については、リファレンス・ページを参照してください。

## 4.2 識別サポート・ライブラリ

Tru64 UNIX オペレーティング・システムでは、ユーザ ID およびグループ ID を管理するためのライブラリ・ルーチンをいくつか提供しています。たとえば、エンハンスド・セキュリティで使用される一部のルーチンでは、`set_auth_parameters()` ルーチンを必要とします。このルーチンは、他のルーチンが後から照会またはテストできるように、初期ユーザ ID および初期グループ ID を格納します。エンハンスド・セキュリティ・オプションを使用するプログラムまたはルーチンを書く場合は、プログラムの `main()` ルーチンの先頭で `set_auth_parameters()` を呼び出さなければなりません。

認証データベースを照会する一部のエンハンスド・セキュリティ・ルーチンを使用するときは、プログラムでユーザ ID またはグループ ID のどれか、またはコマンド引数 `argc` および `argv` を変更する前に、`set_auth_parameters()` を呼び出す必要があります。

詳細については、リファレンス・ページの `identity(3)` を参照してください。

セキュリティ・メカニズム間でのコードの移植性を確保するには、SIA セッション・ルーチンを使用します。

## 4.3 デーモンの使用

デーモンがユーザのプログラム (クライアント) からの要求で操作を実行するときは、次の 2 つのいずれかの方法で動作します。

- 要求元のプログラムが行ってよい動作と行ってはならない動作を自身で判断しながら、自身の ID, アクセス権, および特権の下で動作します。この場合、自身のユーザ ID を変更する必要はありません。

- デーモンがクライアントのセキュリティ属性 (ユーザ ID, アクセス権など) を持ち, 基礎となるオペレーティング・システムの強制動作を利用して実行します。

後者の方法の場合, デーモンは一連のセキュリティ属性を設定する必要があります。お勧めする方法は, プロセスをフォークし, SIA を使って ID と特権を設定し, そして動作を直接行うか, その動作を行うプログラムを実行することです。

## 4.4 エンハンスト (保護) パスワード・データベースの使用

エンハンスト (保護) パスワード・データベースは主に Tru64 UNIX プログラム用ですが, 以下にリストしたフィールドをユーザのプログラムで使用しなければならないこともあります。(これらのフィールドについては, `getespwent(3)` および `prpasswd(4)` のリファレンス・ページ, `prot.h` インクルード・ファイル, および『セキュリティ管理ガイド』でも説明されています。

- ユーザ名 (`u_name`) とユーザ ID (`u_id`) — これらのフィールドは, `/etc/passwd` のユーザ名とユーザ ID に対応します。
- 暗号化パスワード (`u_pwd`) — このフィールドは, 実際の暗号化パスワードです。
- 回収状態 (`u_retired`) — このフィールドは, 認証プロファイルが有効かどうかを示します。有効でない場合は, ログインすることができません。一度, 回収状態になると, そのアカウントを再度使用することはできません。
- ログイン・セッション優先順位 (`u_priority`) — `setpriority()` を使ってユーザのログイン・セッションのプログラムに割り当てられた, プロセスの優先順位です。
- ユーザ監査マスク (`u_auditmask`) と制御フラグ (`u_audcntl`) — このマスクと制御フラグをシステム監査マスクとともに使用して, ログイン・セッションの間に監査されるイベントが決定されます。`login` プログラムは, ユーザのログイン・シェルにマスクを割り当てます。監査マスクと制御フラグは, `exec()` および `fork()` の呼び出し後も継承されます。詳細については, `audcntl(8)` を参照してください。
- パスワード・パラメータ — 以下のパラメータによって, ログイン・パスワードとその生成方法が決まります。

- ユーザが選択したパスワードの最大文字数 (u\_maxchosen)
- パスワード有効期間 (u\_exp)
- 最短のパスワード存続期間 (u\_minchg)
- パスワード存続期間 (u\_life)
- パスワードの変更が成功した最後の日時 (u\_succhg)
- パスワードの変更に失敗した最後の日時 (u\_unsucchg)
- パスワードを最後に変更したユーザ (u\_pwchanger)
- パスワード生成パラメータ (u\_genpwd)
- ユーザが生成したパスワード生成パラメータ (u\_pickpw)
- ログイン・パスワードの必要条件 (u\_nullpw) — 「ヌル・パスワード・オプション」と呼ばれることもあり、空のパスワード設定を制御します。大半の管理者は、このオプションを許可しません。
- ユーザがログインできる時間帯 (u\_tod) — このフィールドのフォーマットは、UUCP の `systems` ファイルに似ています。(systems ファイルには、リモート・システムがファイル転送のために接続できる時間帯を記述します)。このフィールドは、ユーザがログインできる時間帯を決めます。
- 最後にログインした日時 (u\_suclog) — 標準 UNIX 時間 (1970 年からの秒数) で表されます。
- 最後にログインしたときに使用した端末 (u\_suctty) — 端末名は、デバイス割り当てデータベースおよび端末制御データベースとの対応付けに使用されます。
- 最後にログインした後にログインに失敗した回数 (u\_numunsuclog) — この値を使用して、失敗回数が多すぎるために端末を使用不能にするかどうか判断されます。
- 失敗しても使用不能にしないログイン失敗の回数 (u\_maxtries) — この値は、アカウントを使用不能にする前に何回失敗を許すかを示す、ユーザ指定の制限です。
- ロック状態 (u\_lock) — 管理者がアカウントをロックしたかどうかを示します。ロックされたプロファイルは、ログインやその他のサービスには使用できません。システム管理者が明示的に要求したときのみ認証プ

ロファイルがアンロックされ、このような要求を処理するプログラムだけが、ロックされたフィールドをリセットします。よくあるプログラミング上の誤りとして、このロックがすべてのロック状態を示すとみなしてしまうことがあります。これは、管理上のロック状態を示しているだけです。パスワードの存続期間を過ぎたか、アカウントに許可されている失敗回数を越えたために使用不能となり、アカウントがロックされたように見えることがあります。

ユーザのプログラムでは、エンハンスト・セキュリティが有効な場合、エンハンスト (保護) パスワード・データベース内のユーザ名とユーザ ID はシステムが保守し、対応するエントリが `/etc/passwd` ファイルに入っているとみなすことができます。

#### 4.4.1 例:パスワードの有効期限出力プログラム

例 4-1 の `myexpire` という名前のプログラムは、エンハンスト・セキュリティとともに使用するプログラムで、エンハンスト (保護) パスワード・データベースに設定されている、ユーザのパスワード有効期限を出力します。このプログラムは、認証による保護サブシステムの一部で、GID に `auth` を設定するセット・グループ ID (SGID) モードで実行されます。

##### 例 4-1: パスワードの有効期限出力プログラム

---

```
#include <sys/types.h>
#include <stdio.h>
#include <sys/security.h>
#include <prot.h>

main (argc, argv)
int argc;
char *argv[];
{
    struct es_passwd *acct;
    time_t expire_time;
    time_t expire_date;

    /*--- Standard initialization ---*/

    set_auth_parameters(argc, argv);
    initprivs();

    /*--- fetch account information using audit ID ---*/

    if ((acct = getespwuid(getluid())) == NULL)
```

#### 例 4-1: パスワードの有効期限出力プログラム (続き)

---

```
        errmsg("Internal error");

/*-- test if personal or system default applies and print --*/

    if (acct->uflg->fg_expire)
        expire_time = acct->ufld->fd_expire;
    else if (acct->sflg->fg_expire)
        expire_time = acct->sfld->fd_expire;
    else {
        audit_db_error(acct);      /* audit (externally defined) */
        errmsg("No user-specific or system default \
                expiration time.");
    }

    if (!acct->ufld->fg_schange) {
        audit_db_error(acct);      /* audit (externally defined) */
        errmsg("Account does not have successful change time");
    }

    expire_date = acct->ufld->fd_schange + expire_time;

    if (acct->uflg->fg_psw_chg_reqd && \
        acct->ufld->fd_psw_chg_reqd) \
        expire_date = time((time_t *) NULL);

    audit_action(acct->ufld->fd_name, expire_date);
    exit(0);
}
```

---

#### 注意

エンハンスト (保護) パスワード・データベースのファイルは、auth グループのプロセスだけがアクセスできます。エンハンスト・パスワード・データベースのファイルを読み取る必要のあるプログラムでは、グループ ID に auth を設定しなければなりません。setgid(2) のリファレンス・ページを参照してください。この情報を書き込むには、UID に 0 を設定するか、UID にユーザ ID を設定しグループ ID を auth にしなければなりません。

---

---

## 監査レコードの生成

この章では、次の内容について説明しています。

- 監査レコードの概要
- 監査イベント
- 監査レコードとトークン
- 監査フラグとマスク
- 現プロセスの監査の無効化
- 現プロセスの監査の変更
- アプリケーション固有の監査レコード
- サイト定義イベント
- 独自の監査ログの作成
- 監査ログの解析

### 5.1 監査レコードの概要

トラステッド・プログラムは、以下の理由から独自の監査レコードを生成することがあります。

- アプリケーション・レベルの監査を行うことによって、システム・レベルの監査レコードが大量に生成されるのを防ぐ。
- 監査レポートを確認するときに、システム・レベルの監査レコードからだけでユーザの意図を推量するのは難しい。

トラステッド・プログラムは、`audgen()` システム・コール、`audgenl()` ライブラリ・ルーチン、または `audgen` コマンドを使って監査レコードを生成できます。`audgenl()` は、`audgen()` のフロント・エンドです。このプログラムは引数として、監査イベントと、その後に監査データ (監査トークンと値からなる) を渡します。

次のコードでは、ブート認証をチェックするプログラムから `audgenl()` を呼び出して認証の失敗を監査する方法を示しています。

```
if(audgenl(AUTH_EVENT, [1]
           AUD_T_LOGIN, pr->uflld.fd_name, [2]
           AUD_T_UID, pr->uflld.fd_uid,
           AUD_T_CHARP, "boot authentication failed"),0)== -1)
perror("audgenl");
```

注：

- [1] `AUTH_EVENT` は、レコードのイベント名です。
- [2] `AUD_T_LOGIN`、`AUD_T_UID`、および `AUD_T_CHARP` は、トークンです。それぞれ、対応する値を持っています。

これらの識別子は、`<sys/audit.h>` に定義されています。イベントとトークンの説明は、5.2 節および 5.3 節を参照してください。

## 5.2 監査イベント

各監査レコードには、それに対応する監査イベントがあります。システムは、システム・コールの監査レコードを生成するときに、自動的にイベントを追加します。自己監査アプリケーション・プログラムは、監査レコードを生成するときに、イベントを `audgen()` または `audgenl()` に引数として渡します。アプリケーション・プログラムで利用できる監査イベントには、2 つのタイプがあります。

- **トラステッド・イベント。** `MIN_TRUSTED_EVENT` ~ `(MIN_TRUSTED_EVENT + N_TRUSTED_EVENTS - 1)` の値で、`<sys/audit.h>` に定義されているイベントです。たとえば、`LOGIN` イベントです。
- **サイト定義イベント。** `MIN_SITE_EVENT` ~ `1048576` の値で、`/etc/sec/audit_events` に定義されています。サイト定義イベントの省略時の範囲は 64 です。サイト・イベントの定義については、5.8 節を参照してください。

## 5.3 監査レコードとトークン

監査サブシステムでは、固定レコード・タイプは使用しません。監査レコードは、タプル (2 つ以上の構成要素があるデータ・オブジェクト) が連続していま



す。各タプルは、監査トークンとそれに対応する値で構成されます。トークンのタイプによっては、タプルに長さフィールドが存在することもあります。

以降の項では、2つのタイプのトークン (パブリック・トークン と プライベート・トークン) について説明しています。アプリケーション・プログラムでは、パブリック・トークンを使います。

### 5.3.1 パブリック・トークン

`audgen()` および `audgenl()` を使って監査レコードを生成するアプリケーション・プログラムでは、パブリック・トークンを利用できます。パブリック・トークンは、`<sys/audit.h>` に定義されており、`AUD_T_` で始まります。たとえば、`AUD_T_CHARP` です。

パブリック・トークンには3つの基本タイプがあります。

ポインタ	ポインタとしてデータ文字列または構造体を表現するために使います。 <code>AUD_T_CHARP</code> (文字列) と <code>AUD_T_HOMEDIR</code> (ホーム・ディレクトリ) はどちらも、ポインタ・タイプのトークンです。
<code>iovec</code>	データを <code>iovec</code> フォーマットのデータとして表現するために使います。 <code>AUD_T_OPAQUE</code> と <code>AUD_T_INTARRAY</code> はどちらも、 <code>iovec</code> タイプのトークンです。 <code>&lt;sys/audit.h&gt;</code> 内の <code>iovec</code> のコメントを参照してください。 <code>iovec</code> 構造体は、 <code>&lt;sys/uio.h&gt;</code> に定義されています。 <code>iovec</code> についての詳細は、 <code>readv(2)</code> および <code>writew(2)</code> のリファレンス・ページを参照してください。
固定長	データを 32 ビットまたは 64 ビットの量として表現するために使います ( <code>AUD_T_RESULT</code> と <code>AUD_TP_LONG</code> は、64 ビットです。その他は、32 ビットです)。ほとんどのトークンでは、固定長データを使います。 <code>AUD_T_AUDID</code> (監査 ID)、 <code>AUD_T_UID</code> (ユーザ ID)、および <code>AUD_T_PID</code> (プロセス ID) は、固定長トークンの例です。

次の例では、`iovec` フォーマットのデータを使って監査レコードを生成します。

```
#define AUD_COMPAT
#include <sys/audit.h>
#include <sys/uio.h>

main()
{
    char buf[100];
    int i;
    struct iovec iov;

    for (i = 0; i < sizeof(buf); i++)
        buf[i] = i;

    iov.iov_len = sizeof(buf);
    iov.iov_base = buf;

    if (audgenl (AUDGEN8,
                AUD_T_CHARP, "opaque data test",
                AUD_T_OPAQUE, &iov,
                0 ) == -1 )
        perror ("audgenl");
}
```

### 5.3.2 プライベート・トークン

プライベート・トークンは、カーネルが使います。アプリケーション・プログラムでは、このトークンは使用できません。`audgen()` システム・コールは、プライベート・トークンを持つレコードにアプリケーション・プログラムが書き込もうとすると拒否します。プライベート・トークンは、`<sys/audit.h>` に定義されており、`AUD_TP_` で始まります。たとえば、`AUD_TP_AUID` です。

カーネルは、監査レコードを作成するときにプライベート・トークンを使います。たとえばカーネルは、各監査レコードを `AUD_TP_LENGTH` タプル (値がその監査レコードの長さを示す) でカプセル化します。別の例としては、`audgen()` または `audgenl()` の引数 *event* があります。カーネルはこの引数から、`AUD_TP_EVENT` タプルを作成します。

## 5.4 監査フラグとマスク

監査イベントによって実際に監査レコードが生成されるかどうかは、次のフラグとマスクの設定で決まります。

### 5-4 監査レコードの生成

- プロセス監査制御フラグ
- プロセス監査マスク
- システム監査マスク

プロセス監査制御フラグには、4 つの排他的な状態があります。

**AUDIT\_OR**                      システム監査マスクとプロセス監査マスクのいずれかで監査が必要と示されたイベントの場合、監査レコードが生成されます。

**AUDIT\_AND**                    システム監査マスクとプロセス監査マスクの両方で監査が必要と示されたイベントの場合、監査レコードが生成されます。

**AUDIT\_OFF**                    現プロセスに対して監査レコードは生成されません。

**AUDIT\_USR**                    プロセス監査マスクで監査が必要と示されたイベントの場合、監査レコードが生成されます。

また、プロセス監査制御フラグには、排他的でない状態が2 つあります。

**AUDIT\_SYSCALL\_OFF**            プロセスに対するシステム・コールのレコード生成をオフにします。

**AUDIT\_HABITAT\_USR**            システム・マスクでシステム・コールがオフになっていても、プロセスのユーザ・マスク内のハビタット・システム・コールをオンにします。ハビタット・システム・コールには、`System V – unlink()` と `open()`、`real time – memlk()`、`memunlk()`、`psx4_time_drift()`、`rt_setprio()` があります。これらのハビタット・システム・コールは、グループとしてオン、オフされます。ハビタット・イベントについては、付録 B を参照してください。

システム管理者は、管理者が適切と判断するレベルで個々のユーザを監査する機能を残したまま、ユーザに対して省略時の監査レベルを指定することができます (監査サブシステムの構成と管理については『セキュリティ管理ガイド』を参照してください)。

プログラマから見ると、特権プロセスがその監査レベル (監査対象の指定) を設定できます。この監査レベルは、絶対マスクとして設定するか、システム監査マスクとの関連で設定します。プロセスの監査マスクの設定方法を示す例は、5.6 節を参照してください。詳細については、`audcntl(2)` および `auditmask(8)` を参照してください。

## 5.5 現プロセスのシステム・コール監査の無効化

どのイベントを監査するかを制御することは、監査データ収集の量を適切にチューニングするための重要なステップです。システム・コールは、大量の監査データを生成する可能性があります。しかし、このデータは、必ずしも役立つ情報とは限りません。一般的に、セキュリティ関連のデータベース内のフィールドの変更を積極的に監査したり、特定のセキュリティ関連アクションを監査すると、多数のシステム・コール監査レコードから得られる情報よりも、役立つ情報を入手できます。たとえば、ログイン・プロセスはたくさんのシステム・コールを実行しますが、ログイン・プロセスが情報として書き込む 1 つの監査レコードの方が、使用するシステム・リソースが少なく、簡単に理解することができます。

アプリケーション・プログラムは、トラステッド・イベントの監査を許可したまま、システム・コール監査を無効にすることができます。次のコードでは、`audcntl()` システム・コールを使って `AUDIT_SYSCALL_OFF` を設定する方法を示しています。

```
/* OR the AUDIT_SYSCALL_OFF bit into the audcntl flag */
if ((cntlflag = audcntl(GET_PROC_ACNTL,
                        NULL, 0, 0, 0, 0)) == -1)
    perror("audcntl");
else
    audcntl(SET_PROC_ACNTL, NULL, 0,
           cntlflag|AUDIT_SYSCALL_OFF, 0, 0);
```

## 5.6 現プロセスのシステム・コールの監査の変更

プロセスは、ターゲット・プロセスの `auditmask` フラグと `audcntl` フラグを変更することによって、プロセス自身または別のプロセスに対して何

を監査するかを制御できます。次のように、現プロセスの監査マスクを変更できます。

```
/* ex. set the process's auditmask to audit only LOGIN
   events and successful setgroups calls
*/
#include <sys/audit.h>
#include <sys/syscall.h>
char buf[AUDIT_MASK_LEN];
:
:
bzero (buf, sizeof(buf));
A_PROCMASK_SET (buf, LOGIN, 1, 1);
A_PROCMASK_SET (buf, SYS_setgroups, 1, 0);
if (audcntl (SET_PROC_AMASK, buf,
             AUDIT_MASK_LEN, 0, 0, 0) == -1)
    perror ("audcntl");
```

<sys/audit.h> に定義されている A\_PROCMASK\_SET マクロは、次の引数を取ります。

<i>buf</i>	マスクが設定されているバッファです。
<i>event name</i>	ヘッダ・ファイル <sys/audit.h> には、トラステッド・イベント名があります。ヘッダ・ファイル <sys/*syscall.h> には、システム・コール名があります。
<i>succeed</i>	成功を監査するかどうかを示します。1 は、イベントの成功を監査することを意味します。
<i>fail</i>	失敗を監査するかどうかを示します。1 は、イベントの失敗を監査することを意味します。

詳細については、audcntl(2) を参照してください。

## 5.7 アプリケーション固有の監査レコード

アプリケーション・プログラムは、アプリケーション固有の監査データを audgen() または audgenl() への引数として渡します。

次のコードは、指定された *event* が発生したときに、監査レコードをカーネルに送信します。*event* は、<sys/audit.h> からのトラステッド・イベントか、/etc/sec/site\_events からのサイト定義イベントです。(カーネル

が監査レコードを監査ログに実際に書き込むかどうかは、このプロセスで監査されているイベントによります)。

```
/* If bad_thing occurs, generate an event of type event_num,
 * with string "bad thing happened", and a result of 66.
 */

#include <sys/audit.h>

if (bad_thing) {
    if (audgenl (event_num,
                AUD_T_CHARP, "bad thing happened",
                AUD_T_RESULT, 66, 0 ) == -1)
        perror ("audgenl");
}
```

通常、アプリケーションが生成する監査レコードには、表 5-1 に示すトークンのデータを入れる必要はありません。カーネルは、この情報を各監査レコードに自動的に追加します。ただし監査サブシステムは、監査レコードにパブリック・トークンのタプルが追加されるのを妨げません。たとえば、システムが後から AUD\_TP\_AUDID を監査レコードに追加する場合でも、AUD\_T\_AUDID タプルを監査レコードに追加できます。このような場合は両方のタプルが監査ログに書き込まれます。

## 5.8 サイト定義イベント

サイトでは、独自の監査イベントの設定(サイト定義イベント)を、ローカルに作成され保守されるファイル /etc/sec/site\_events で定義できます。このファイル内には、各サイト・イベントに対するエントリが1つあります。

使用可能なサイト・イベント番号は、MIN\_SITE\_EVENT (<sys/audit.h> に定義されている) ~ 1048576 です。省略時の範囲は、64 です。この値を変更するには、/etc/sysconfigtab 内の audit-site-events を設定してから、リブートします。たとえば、サイト定義イベントを最大 128 個にするには、次のように設定します。

```
sec:
    audit-site-events=128
```

各サイト・イベント・エントリは、INT\_MAX 個までのサブイベントを持つことができます。サブイベントには、省略時の範囲は定義されていません。

イベントまたはサブイベントの名前の最大長は、<sys/audit.h> に定義されている AUD\_MAXEVENT\_LEN です。

### 5-8 監査レコードの生成

アプリケーション・プログラムは、サイト定義イベントと、`<sys/audit.h>` に定義されているトラステッド・イベント (`MIN_TRUSTED_EVENT` ~ `MAX_TRUSTED_EVENT`) の両方を持つレコードを生成できます。

`auditmask` ユーティリティは、サイト定義イベントの事前選択をサポートしています。また、`audit_tool` ユーティリティは、サイト定義イベントとサブイベントの事後選択をサポートしています。

### 5.8.1 サンプルの `site_events` ファイル

サイト定義監査イベント・エントリの構文は次のとおりです。

[ イベント名 イベント番号 [, サブイベント名 サブイベント番号 ...] ;]

サンプルの `/etc/sec/site_events` ファイルの次のエントリは、サイト定義イベントおよびサブイベントの作成方法を示しています。

```
essence 2048, 1  
    ess_read 0, 2  
    ess_write 1; 3  
rdb 2049,  
    rdb_open 0,  
    rdb_close 1,  
    rdb_read 2,  
    rdb_write 3;  
decinspect 2050;
```

注：

- 1 `essence` はイベントです。2048 はイベント番号です。2048 は `MIN_SITE_EVENT` で、サイト定義イベントに使用できる最小値です。
- 2 `ess_read` は 1 番目のサブイベントです。0 は 1 番目のサブイベント番号です。
- 3 `ess_write` は 2 番目のサブイベントです。1 は 2 番目のサブイベント番号です。

サイト定義イベントについての詳細は、`aud_siteevent(3)` を参照してください。

### 5.8.2 例: サイト定義監査イベントの監査レコードの生成

次のコードは、`audgenl()` を使って、`rdb_close` イベントの監査データを生成します。

```

int event_num, subevent_num;

/* translate event name(s) into event numbers */
if (aud_siteevent_num ("rdb", "rdb_close",
                      &event_num, &subevent_num ))
    printf ("aud_siteevent_num failed");

/* generate audit data */
else if (audgenl (event_num,
                 AUD_T_SUBEVENT, subevent_num,
                 AUD_T_CHARP, "Trusted RDB V1.0 Close",
                 0) == -1)
    perror ("audgenl");

```

サイト定義イベントのレコードを生成する場合は、`audgenl()` に `AUD_T_CHARP, event name` という引数を含めてください。これにより、ローカルの `site_events` ファイルのコピーを持たないシステムで監査データを解析する作業が簡単になります。

詳細については、`aud_siteevent(3)` および `audgenl(3)` を参照してください。

## 5.9 独自の監査ログの作成

`audgen()` システム・コールを使って、独自の監査ログを作成できます。`audgen()` への引数 `size` がゼロ以外の値の場合、監査データは、システム監査ログに書き込まれるのではなく、`audgen()` で指定した `userbuff` へコピーされます。この後、トラステッド・アプリケーションは、`userbuff` のデータを固有のログ・ファイルに書き込むことができます。詳細については、`audgen(2)` を参照してください。

`audit_tool` ユーティリティを使って、新しい監査ログを読み取ることができます。5.10 節の説明を参考に、ログから詳しい情報を読み取ることができます。

## 5.10 監査ログの解析

多くの人が、`audit_tool` または `dxaudit` を使用して監査ログを参照します。`audit_tool` ユーティリティは洗練されたプログラムで、監査データを役立つ情報に変換したり、出力をフォーマットしたり、複数の監査ログ・ファイルにまたがる監査レコードを処理します。`audit_tool` は、監査ログを最初に読み取ったとき、そのログに対応する `.hdr` ファイルを作成して、状態情報を保守します。この状態情報により、その後の監査ログの読み取りに必要な時間が短縮されます。また、監査レコードが複数のログにま



たがる場合は、audit\_tool が両方のログ・ファイルをオープンし、ヘッダ・ファイル内に完全なレコードを作成します。

以降の項では、監査ログのフォーマットと構造について、次のような情報を説明しています。

- 監査ログの説明と、すべての監査レコードにある一般的なトークン・タイプのリスト
- バイナリ・レコードのフォーマットと、レコードの8進ダンプおよびそのフォーマットされた出力を示す例
- パブリック・トークンおよびプライベート・トークンのデータ・タイプおよびフォーマットをリストした、トークンおよびタブルのバイト記述についての表
- タブル解析のためのサンプル・マクロ

以降の項では、audit\_tool のようなプログラムを作成するために必要な、設計に関する情報は説明していません。監査ログを解析してレコードとタブルに分解するために必要な基本情報について説明しています。

5.10.1 監査ログのフォーマットの概要と共通タブルのリスト

監査ログは、通常の UNIX データ・ファイルで、監査レコードが格納されています。監査レコードは、フォーマットが *token:value* または *token:length:value* のどちらかの、連続したタブルで構成されています。各レコードは、AUD\_TP\_LENGTH タブルで始まり、AUD\_TP\_LENGTH タブルで終わります。audit\_tool ユーティリティは、監査レコードが有効かどうかを判断するのに AUD\_TP\_LENGTH を使います。レコードの実際の長さが AUD\_TP\_LENGTH の値と一致しない場合は、audit\_tool がレコードを破棄し、警告を出力します。表 5-1 は、監査レコードで広く使われている省略時のタブルを示しています。

表 5-1: 大半の監査レコードに共通する省略時のタブル

タブル	コメント	タブル	コメント
AUD_TP_LENGTH		AUD_TP_VERSION	
AUD_TP_AUDID		AUD_TP_RUID	
AUD_TP_HOSTADDR		AUD_TP_EVENTP	ハビタットの場合

表 5-1: 大半の監査レコードに共通する省略時のタプル (続き)

タプル	コメント	タプル	コメント
AUD_TP_HABITAT	ハビタット の場合	AUD_TP_EVENT	
AUD_TP_UID		AUD_TP_PID	
AUD_TP_PPID		AUD_TP_DEV	デバイスがプ ロセスに対応 している場合
AUD_TP_NCPU	uid の変更 の場合	AUD_TP_TV_USEC	
AUD_TP_SET_UIDS		AUD_TP_TID	AUDIT_USR フラグが設 定されてい る場合

5.10.2 トークンおよびタプルのバイト記述

表 5-2 は、パブリック・トークンとプライベート・トークンを、8 進値と一緒に示しています。各タプルの 3 番目の欄は、カーネルが監査ログに書き込むタプル・データのシーケンスを示しています。トークンは 1 バイト、長さは 4 バイトです。サンプルの解析マクロは、`audit_tool` がタプルを解析するために使うマクロを示しています。参考として、これらのマクロを 5.10.3 項に記載します。

表 5-2: トークンおよびタプルのバイト記述

トークン	8 進値	タプルのフォーマットとサンプルの解析マクロ
AUD_T_CHARP	001	トークン:長さ:ヌル終了する ASCII 文字列。PARSE_DEF3
AUD_T_SOCK	003	トークン:長さ:struct sockaddr (4.3 スタイル (u_short)。family > UCHAR_MAX の場合は 4.4 スタイルの sockaddr となり、「長さ」(バイト)の後に family (バイト)が続く)。PARSE_DEF3
AUD_T_LOGIN	004	トークン:長さ:ヌル終了する ASCII 文字列。PARSE_DEF3
AUD_T_HOMEDIR	005	トークン:長さ:ヌル終了する ASCII 文字列。PARSE_DEF3
AUD_T_SHELL	006	トークン:長さ:ヌル終了する ASCII 文字列。PARSE_DEF3

表 5-2: トークンおよびタプルのバイト記述 (続き)

トークン	8 進値	タプルのフォーマットとサンプルの解析マクロ
AUD_T_DEVNAME	007	トークン:長さ:ヌル終了する ASCII 文字列。PARSE_DEF3
AUD_T_SERVICE	010	トークン:長さ:ヌル終了する ASCII 文字列。(将来の使用のために予約済み)
AUD_T_HOSTNAME	011	トークン:長さ:ヌル終了する ASCII 文字列。PARSE_DEF3
AUD_T_INTP	012	トークン:長さ:int (1 番目の要素は配列内の要素の数。監査レコードの生成時には、AUD_T_INTARRAY の方が適切なタプルである。PARSE_DEF3
AUD_T_LSOCK	016	
AUD_T_RSOCK	017	
AUD_T_LHOSTNAME	020	
AUD_T_OPAQUE	030	トークン:長さ:値 (proplist または本当の opaque。proplist の名前と値の組をチェックし、なければ 16 進でダンプ)。PARSE_DEF6
AUD_T_INTARRAY	031	トークン:長さ:int。PARSE_DEF3
AUD_T_GIDSET	032	トークン:長さ:int1, int2, ... (境界合わせなし)。PARSE_DEF3
AUD_T_XDATA	033	トークン:struct aud_xdata (<sys/audit.h>を参照)。PARSE_DEF8
AUD_T_AUDID	040	トークン:int。PARSE_DEF2
AUD_T_RUID	041	トークン:int。PARSE_DEF2
AUD_T_UID	042	トークン:int。PARSE_DEF2
AUD_T_PID	043	トークン:int。PARSE_DEF2
AUD_T_PPID	044	トークン:int。PARSE_DEF2
AUD_T_GID	045	トークン:unsigned int。PARSE_DEF2
AUD_T_EVENT	046	トークン:int。PARSE_DEF2
AUD_T_SUBEVENT	047	トークン:int。PARSE_DEF2
AUD_T_DEV	050	トークン:int (<sys/types.h> のマクロ major() および minor() を使って解析する)。PARSE_DEF2

表 5-2: トークンおよびタブルのバイト記述 (続き)

トークン	8 進値	タブルのフォーマットとサンプルの解析マクロ
AUD_T_ERRNO	051	トークン:int。PARSE_DEF1
AUD_T_RESULT	052	トークン:long。PARSE_DEF4
AUD_T_MODE	053	トークン:unsigned int。PARSE_DEF2
AUD_T_HOSTADDR	054	トークン:unsigned int。PARSE_DEF2
AUD_T_INT	055	トークン:int。PARSE_DEF2
AUD_T_DESCRIP	056	トークン:int (ファイル記述子)。PARSE_DEF2
AUD_T_HOSTID	057	トークン:int。PARSE_DEF1
AUD_T_X_ATOM	060	トークン:unsigned int。PARSE_DEF2
AUD_T_X_CLIENT	061	トークン:int。PARSE_DEF2
AUD_T_X_PROPERTY	062	トークン:int。PARSE_DEF2
AUD_T_X_RES_CLASS	063	トークン:unsigned int。PARSE_DEF2
AUD_T_X_RES_TYPE	064	トークン:unsigned int。PARSE_DEF2
AUD_T_X_RES_ID	065	トークン:unsigned int。PARSE_DEF2
AUD_T_LHOSTNAME	066	
AUD_T_SECEVENT	177	トークン:int。PARSE_DEF2
AUD_TP_ACCRGHT	201	トークン:長さ:cmsg_data (fd1, fd2, ... 。 <sys/socket.h> を参照)。PARSE_DEF3
AUD_TP_MSGHDR	202	トークン:長さ:msg_hdr->msg_name (<sys/socket.h> を参照)。PARSE_DEF3
AUD_TP_EVENTP	203	トークン:長さ:string。PARSE_DEF3
AUD_TP_HABITAT	204	トークン:長さ:string。PARSE_DEF3
AUD_TP_ADDRVEC	205	トークン:長さ:struct sockaddr (socket.h を参照)。PARSE_DEF3
AUD_TP_INTP	206	トークン:長さ:int。PARSE_DEF3
AUD_TP_AUID	241	トークン:int。PARSE_DEF1
AUD_TP_RUID	0242	トークン:int。PARSE_DEF1
AUD_TP_UID	0243	トークン:int。PARSE_DEF1
AUD_TP_PID	0244	トークン:int。PARSE_DEF1

表 5-2: トークンおよびタプルのバイト記述 (続き)

トークン	8 進値	タプルのフォーマットとサンプルの解析マクロ
AUD_TP_PPID	0245	トークン:int。PARSE_DEF1
AUD_TP_HOSTADDR	246	トークン:unsigned int。PARSE_DEF1
AUD_TP_EVENT	247	トークン:int。PARSE_DEF1
AUD_TP_SUBEVENT	250	トークン:int (将来の使用のために予約済み)。PARSE_DEF1
AUD_TP_NCPU	251	トークン:int。PARSE_DEF1
AUD_TP_DEV	252	トークン:int (sys/types.h のマクロ major() および minor() を使って解析する)。PARSE_DEF1
AUD_TP_LENGTH	253	トークン:int。PARSE_DEF1
AUD_TP_IPC_GID	254	トークン:unsigned int (ipc msg shm_perm.gid)。PARSE_DEF2
AUD_TP_IPC_MODE	255	トークン:unsigned int (ipc msg shm_perm.mode)。PARSE_DEF2
AUD_TP_IPC_UID	256	トークン:int (ipc msg shm_perm.uid)。PARSE_DEF2
AUD_TP_TV_SEC	257	トークン:timeval.tv_sec (<sys/time.h> を参照)。PARSE_DEF1
AUD_TP_TV_USEC	260	トークン:timeval.tv_usec (<sys/time.h> を参照)。PARSE_DEF1
AUD_TP_SHORT	261	トークン:short。PARSE_DEF2
AUD_TP_LONG	262	トークン:long。PARSE_DEF5
AUD_TP_VNODE_DEV	263	トークン:int。PARSE_DEF2
AUD_TP_VNODE_ID	264	トークン:unsigned int。PARSE_DEF2
AUD_TP_VN- ODE_MODE	265	トークン:unsigned int。PARSE_DEF2
AUD_TP_VERSION	266	トークン:unsigned int。(<sys/audit.h> を参照)。(AUD_VERSION   AUD_VERS_LONG) PARSE_DEF1
AUD_TP_SET_UIDS	267	トークン:int。PARSE_DEF2
AUD_TP_CONT	270	トークン:unsigned int (レコードの各構成要素に固有の int)。PARSE_DEF1

表 5-2: トークンおよびタブルのバイト記述 (続き)

トークン	8 進値	タブルのフォーマットとサンプルの解析マクロ
AUD_TP_TID	271	トークン:long。PARSE_DEF4
AUD_TP_PRIV	272	トークン:unsigned short。PARSE_DEF1

5.10.3 タブルの解析

監査レコードのストリームを読み取るアルゴリズムは次のとおりです。

1. 監査ログをオープンします。
2. 最初の監査レコードを見つけます (AUD\_TP\_LENGTH タブルで始まり , AUD\_TP\_LENGTH タブルで終わります)。
3. レコード長が AUD\_TP\_LENGTH タブルの値と一致するかチェックします (長さが一致しない場合は , レコードを破棄します)。
4. AUD\_TP\_LENGTH タブルに続く最初のタブルを取り出します。
5. タブルが可変長の場合は , データのサイズを調べます。
6. データを抽出します。
7. 次のタブルを取り出し , 必要に応じて長さをチェックします。そして , データを抽出します。
8. レコードがなくなるまで繰り返します。
9. 監査ログをクローズします。

次のマクロは , audit\_tool がどのようにタブルを解析するかを示しています。これらのマクロは , 参考として記載しているだけで , 1 つの手段を示しているに過ぎません。indx の値は , audit\_tool が保守し , 使用します。この値は , 監査レコードのタブルの一部ではありません。

```
/* fixed length scalar value */
#define PARSE_DEF1(tokentype,field) \
    bcopy (&rec_ptr[i+sizeof token], &field, sizeof(field)); \
    i += (sizeof token + sizeof(field)); \
break;

/* fixed length field in array */
#define PARSE_DEF2(tokentype,field,indx) \
    if (indx < AUD_NPARAM) \
        bcopy (&rec_ptr[i+sizeof token], &field[indx++], sizeof(field[0])); \
    i += (sizeof token + sizeof(field[0])); \
break;

/* array of strings */
```

```

#define PARSE_DEF3(tokentype,len,field,indx) \
    bcopy (&rec_ptr[i+sizeof token], &j, sizeof(int)); \
    if (j >= rec_len) j = 0; \
    if (indx < AUD_NPARAM) { \
        len[indx] = j; \
        field[indx++] = (char *)&rec_ptr[i+(sizeof token)+(sizeof *intp)]; \
    } \
    i += (sizeof token + sizeof *intp + j); \
break;

/* fixed length scalar value whose size is h/w dependent (32 or 64-bit) */
#define PARSE_DEF4(tokentype,field) \
    bzero (field.val, sizeof(field.val)); \
    j = af->version & AUD_VERS_LONG ? sizeof(int)*2 : sizeof(int); \
    bcopy (&rec_ptr[i+sizeof token], field.val, j); \
    i += (sizeof token + j); \
break;

/* fixed length field in array whose size is h/w dependent (32 or 64-bit) */
#define PARSE_DEF5(tokentype,field,indx) \
    bzero (field[indx].val, sizeof(field[indx].val)); \
    j = af->version & AUD_VERS_LONG ? sizeof(int)*2 : sizeof(int); \
    if (indx < AUD_NPARAM) \
        bcopy (&rec_ptr[i+sizeof token], field[indx++].val, j); \
    i += (sizeof token + j); \
break;

/* array of opaque data streams */
#define PARSE_DEF6 PARSE_DEF3

/* iovec element in array */
#define PARSE_DEF7(tokentype,field,indx) \
    j = sizeof(field[0]); \
    if (indx < AUD_NPARAM) { \
        bcopy (&rec_ptr[i+sizeof token], &j, sizeof(int)); \
        if (j > rec_len) j = 0; \
        bcopy (&rec_ptr[i+sizeof token+sizeof(int)], &field[indx++], j); \
    } \
    i += (sizeof token + sizeof(int) + j); \
break;

/* array of iovec elements with variable length components */
#define PARSE_DEF8(tokentype,field,ptr,indx) \
    j = sizeof(field[0]); \
    if (indx < AUD_NPARAM) { \
        bcopy (&rec_ptr[i+sizeof token], &j, sizeof(int)); \
        if (j > rec_len) j = 0; \
        bcopy (&rec_ptr[i+sizeof token+sizeof(int)], &field[indx], j); \
        ptr[indx++] = ((struct aud_xdata *) \
            &rec_ptr[i+sizeof token+sizeof(int)])->xdata; \
    } \
    i += (sizeof token + sizeof(int) + j); \
break;

```





---

## SIA インタフェースの使用

この章では、次の内容について説明しています。

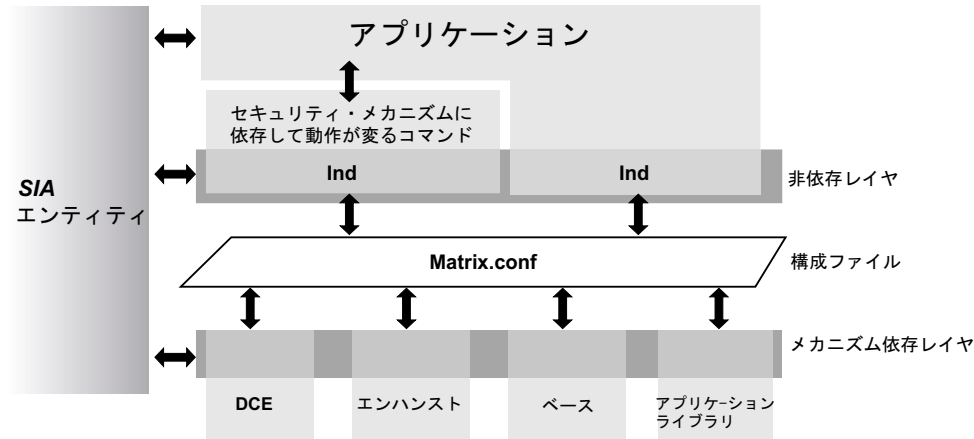
- SIA の概要
- SIA のアーキテクチャ
- SIA システムの初期設定
- SIAENTITY 構造体
- SIA パラメータの収集
- 状態の保守
- SIA のリターン値
- SIA のデバッグとログ
- セキュリティ・メカニズムの統合
- SIA セッション処理
- セキュア情報の変更
- セキュリティ情報へのアクセス
- セッション・パラメータの収集
- SIA 製品のパッケージ
- セキュリティ・メカニズムに依存するインタフェース
- シングルユーザ・モード
- SIA ルーチンのシンボル優先使用

### 6.1 SIA の概要

セキュリティ統合アーキテクチャ (SIA) を使用すると、ローカルおよび分散型のセキュリティ認証メカニズムを Tru64 UNIX オペレーティング・システム上に階層化できます。SIA 構成フレームワークは、セキュリティを扱うコマンドを特定のセキュリティ・メカニズムから切り離します。Tru64 UNIX の

セキュリティを扱うコマンドは、メカニズムに依存するルーチン群を呼び出すために変更されています。固有のルーチン群を持つライブラリを用意することで、開発者は、セキュリティを扱うコマンド自体を変更せずに、そのコマンドの動作を変更することができます。SIA では、SIA 構成に必要な、セキュリティ・メカニズムに依存するインタフェース (siad\_\*() ルーチン) を定義しています。図 6-1 は、SIA を形成する構成要素の関係を示しています。

図 6-1: SIA の階層構造



ZK1086UAI

セキュリティを扱うコマンドを、表 6-1 に示します。

表 6-1: セキュリティを扱うオペレーティング・システム・コマンド

コマンド	説明
chfn	finger 情報を変更します。
chsh	ログイン・シェル情報を変更します。
dnascd	DECnet と接続します。
ftpd	インターネット・ファイル転送プロトコル・サービスを行います。
login	ユーザの認証を行います。
passwd	ユーザ・パスワードを設定または変更します。
rshd	リモート実行のサービスを行います。
su	ユーザ ID を切り替えます。

表 6-2 および表 6-3 は、SIA ポート・ルーチンを示しています。

表 6-2: SIA のメカニズム非依存ルーチン

SIA ルーチン	説明
<code>sia_init()</code>	SIA 構成を初期化します。
<code>sia_chk_invoker()</code>	呼び出し元アプリケーションの特権を チェックします。
<code>sia_collect_trm()</code>	パラメータを収集します。
<code>sia_chg_finger()</code>	<code>finger</code> 情報を変更します。
<code>sia_chg_password()</code>	ユーザのパスワードを変更します。
<code>sia_chg_shell()</code>	ログイン・シェルを変更します。
<code>sia_ses_init()</code>	SIA セッション処理を初期化します。
<code>sia_ses_authent()</code>	エンティティの認証を行います。
<code>sia_ses_reauthent()</code>	ユーザのパスワードを再確認します。
<code>sia_ses_suauthent()</code>	<code>su</code> コマンドを処理します。
<code>sia_ses_estab()</code>	セッションのためのコンテキストを確立します。
<code>sia_ses_launch()</code>	セッション開始および TTY 状態のログをとります。
<code>sia_ses_release()</code>	セッションに関連するリソースを解放します。
<code>sia_make_entity_pwd()</code>	SIAENTITY のパスワード構造体を用意します。
<code>sia_audit()</code>	監査レコードを生成します。
<code>sia_chdir()</code>	現ディレクトリを安全に変更します (NFS セーフ)。
<code>sia_timed_action()</code>	時間制限およびシグナルの保護付きで呼 び出しを行います。
<code>sia_become_user()</code>	<code>su</code> ルーチン。
<code>sia_validate_user()</code>	ユーザのパスワードを確認します。
<code>sia_get_groups()</code>	グループを取得します。

表 6-3: SIA のメカニズム依存ルーチン

SIA ルーチン	説明
<code>siad_init()</code>	処理を初期化します (リブートのたびに呼び出されます)。
<code>siad_chk_invoker()</code>	呼び出し元プログラムの特権をチェックします。
<code>siad_ses_init()</code>	セッションを初期化します。
<code>siad_ses_authent()</code>	セッションの認証を行います。
<code>siad_ses_estab()</code>	リソースとライセンスをチェックします。
<code>siad_ses_launch()</code>	セッション開始のログをとります。
<code>siad_ses_suauthent()</code>	<code>su</code> コマンドを処理します。
<code>siad_ses_reauthent()</code>	ユーザのパスワードを再確認します。
<code>siad_ses_release()</code>	セッションのリソースを解放します。
<code>siad_chg_finger()</code>	<code>chfn</code> コマンドを処理します。
<code>siad_chg_password()</code>	パスワードを変更する関数を呼び出します。
<code>siad_chg_shell()</code>	<code>chsh</code> コマンドを処理します。
<code>siad_getpwent()</code>	<code>getpwent()</code> および <code>getpwent_r()</code> を処理します。
<code>siad_getpwuid()</code>	<code>getpwuid()</code> および <code>getpwuid_r()</code> を処理します。
<code>siad_getpwnam()</code>	<code>getpwnam()</code> および <code>getpwnam_r()</code> を処理します。
<code>siad_setpwent()</code>	一連の <code>getpwent()</code> 呼び出しを初期化します。
<code>siad_endpwent()</code>	一連の <code>getpwent()</code> 呼び出し後にリソースを解放します。
<code>siad_getgrent()</code>	<code>getgrent()</code> および <code>getgrent_r()</code> を処理します。
<code>siad_getgrgid()</code>	<code>getgrgid()</code> および <code>getgrgid_r()</code> を処理します。
<code>siad_getgrnam()</code>	<code>getgrnam()</code> および <code>getgrnam_r()</code> を処理します。
<code>siad_setgrent()</code>	一連の <code>getgrent()</code> 呼び出しを初期化します。
<code>siad_endgrent()</code>	一連の <code>getgrent()</code> 呼び出しをクローズします。
<code>siad_chk_user()</code>	要求された情報をメカニズムが変更できるか確認します。
<code>siad_get_groups()</code>	ユーザの補助グループの配列を埋めます。

SIA は、セキュリティを扱うコマンドと、セキュリティ・メカニズムに依存する関数を提供するセキュリティ・メカニズムの間に、レイヤを確立します。セキュリティを扱う SIA ルーチンは、セキュリティ・メカニズムをそれぞれ 4 つまで使用するように構成できます (呼び出される順序はまちまちです)。

どのセキュリティ・メカニズムをどのような順序で呼び出すかは、スイッチ・テーブル・ファイル `/etc/sia/matrix.conf` で決定されます (『セキュリティ管理ガイド』を参照)。この方法は、`/etc/svc.conf` を使って `libc get*` 関数を制御する方法と似ています。ただし、呼び出しメカニズムは明らかに異なります。

SIA の呼び出しメカニズムは、シェアード・ライブラリ内のルーチンのアドレスを見つけて呼び出し、特定のセキュリティ・メカニズムのルーチンにアクセスします。SIA には、Tru64 UNIX の `getpw*` 関数および `getgr*` 関数の代わりとなる制御および構成が用意されています。

SIA の階層構造により、透過性を必要とする新しいセキュリティ・メカニズムやセキュリティを扱うコマンドに対し、国際化メッセージ・カタログ・サポートおよびスレッド・セーフのポート・インタフェースが確立されます。SIA インタフェースのタイプごとに用意されたロック群によって、スレッド・セーフが実現されています。ただし、SIA はユーティリティとセキュリティ・メカニズムとの間のレイヤであるため、再入可能性を実現するのは、レイヤード・セキュリティ・メカニズムの責任となります。

SIA の主な目的は、将来のセキュリティ要件に対応できるだけ柔軟で拡張性がある、`login`、`su`、および `passwd` などの、セキュリティを扱うコマンド用の透過的なインタフェースを提供することです。Tru64 UNIX 上のレイヤード・プロダクトのうち、新しいセキュリティ・メカニズムを作成するか、セキュリティを扱うコマンドを含む製品には、透過的なインタフェースを維持するために SIA を統合する必要があります。

SIA の構成要素は、ユーザ・レベルのモジュールだけで構成されます。これらの構成要素は、セキュリティを扱うコマンドが複数のセキュリティ・メカニズムを利用するという構成上の問題を解決します。SIA の構成要素は、複数のセキュリティ・メカニズムの構成および利用に伴うカーネル関連の問題は解決しません。

## 6.2 SIA のアーキテクチャ

Tru64 UNIX の SIA で導入された階層型アーキテクチャには、次の 2 つのグループのインタフェース・ルーチンがあります。

`sia_*()`                      セキュリティを扱うコマンドが使う、セキュリティ・メカニズムに依存しないインタフェース

`siad_*()`                      特定のセキュリティ・メカニズムによって提供される、セキュリティ・メカニズムに依存するインタフェース

各セキュリティ・メカニズムでは、`siad_*()` ルーチンを含むシェアード・ライブラリを提供しています。また、構成に合った、固有のセキュリティ・メカニズム名を提供しています。どのメカニズムをどのような順序で呼び出すかを指定するために、1 単語のセキュリティ・メカニズム名とライブラリ名を、`matrix.conf` ファイル内のキーとして使います。

Tru64 UNIX のセキュリティを扱うコマンドは、メカニズムに依存しない `sia_*()` ルーチンを使うように変更されています。これらのルーチンは、特定のセキュリティ技術に依存しないセキュリティ関数にアクセスするために、コマンドとユーティリティで使います。`matrix.conf` ファイルで選択された構成に応じて、各 `sia_*()` ルーチンは、メカニズムに依存する、関連の `siad_*()` ルーチンを呼び出します。このファイルについての詳細は、『セキュリティ管理ガイド』を参照してください。

メカニズムに依存する `siad_*()` インタフェース・ルーチンは、セキュリティ・メカニズムが提供する、メカニズム依存関数へのコールアウトとして、SIA で定義されています。`matrix.conf` ファイルは、各 SIA 関数に対して呼び出されるセキュリティ・メカニズムと、その順序を決定するために使用されます。

指定されたセキュリティ・メカニズム内の特定のモジュールを呼び出し、必要な状態を渡す処理は、メカニズムに依存しない層で行われます。呼び出し元プロセスは、シェアード・ライブラリ関数を使って、セキュリティ・メカニズムで提供される指定のシェアード・ライブラリ内の特定のモジュール・アドレスにアクセスしたり、アドレスを見つけたりします。

名前の衝突をなくし、呼び出し手順を簡単にするために、セキュリティ・メカニズムに依存するモジュール `siad_*()` ルーチンの名前は固定です。Tru64 UNIX は、`dlopen()` および `dlsym()` シェアード・ライブラリ・インタフェースを使って、特定のセキュリティ・メカニズムのシェアード・ライブラリをオープンし、`siad_*()` 関数のアドレスを見つけます。`siad_*()` ルーチンを優先使用 (preempt) する必要がある場合、ライブラリ内で `__siad_*` という形式の名前にしなければなりません。また、そのライブラリを `libc` より先にリンクしなければなりません。命名と優先使用の要件についての詳細は、6.17 節を参照してください。

### 6.2.1 ライブラリ

SIA のセキュリティ・メカニズムは、個別のシェアード・ライブラリとして構成されます。このシェアード・ライブラリのエントリ・ポイントには、SIA で定義された名前を付けます。各メカニズムは、一意のメカニズム識別子を持つ必要があります。セキュリティ・メカニズムで提供される、シェアード・ライブラリ内の実際のエントリ・ポイントは各メカニズムとも同じで、`siad_*()` 形式のエントリ・ポイントです。

省略時のセキュリティ構成は、`libc` に含まれている BASE セキュリティ・メカニズムです。省略時の BASE セキュリティ・メカニズムは、ユーザ・データベースとして `/etc/passwd` ファイル (つまり、ハッシュ・データベース・バージョン)、グループのデータベースとして `/etc/group` ファイルを使います。NIS (Network Information Service) が構成されている場合は、省略時の BASE メカニズムはこのサービスも使います。シングルユーザ・モードのとき、またはインストール時には、BASE セキュリティ・メカニズムが有効になります。

### 6.2.2 ヘッド・ファイル

SIA インタフェースと構造体は、ファイル `/usr/include/sia.h` および `/usr/include/siad.h` に定義されています。`sia*.h` ファイルは、プログラム開発サブセットに含まれています。

## 6.3 SIA システムの初期設定

SIA は、システムがリブートされるたびに各セキュリティ・メカニズムへのコールアウトを用意します。このコールアウトは、`/usr/sbin/siainit` プログラムが実行します。このプログラムは、構成されている各セキュリ

ティ・メカニズムの `siad_init()` エントリ・ポイントを呼び出します。これにより、セキュリティ・メカニズムはリブート時の初期化を行うことができます。`siad_init()` 呼び出しから SIADFAIL 応答があると、システムはリブートしなくなり、SIA INITIALIZATION FAILURE (SIA 初期化障害) メッセージがコンソールへ送信されます。このため、`siad_init()` 呼び出しが SIADFAIL 応答を行うのは、セキュリティ上の問題や、`root` にログインが許可されないという問題だけとする必要があります。

## 6.4 SIAENTITY 構造体

SIAENTITY 構造体には、セッション処理のパラメータが格納されています。これらのパラメータは、セッション処理の段階間でセッションの状態を引き渡すために使用されます。例 6-1 は、SIAENTITY 構造体を示しています。

例 6-1: SIAENTITY 構造体

---

```
typedef struct siaentity {
    char *name;                /* collected name */
    char *password;            /* entered or collected password */
    char *acctname;            /* verified account name */
    char **argv;               /* calling command argument list */
    int argc;                  /* number of arguments */
    uid_t suid;                /* starting ruid */
    char *hostname;            /* requesting host NULL=>local */
    char *tty;                 /* pathname of local tty */
    int can_collect_input;     /* 1 => yes, 0 => no input */
    int error;                  /* error message value */
    int authcount;             /* Number of consecutive
                               /* failed authent attempts
    int authtype;              /* Type of last authent
    struct passwd *pwd;        /* pointer to passwd struct
    char *gssapi;              /* for gss_api prototyping
    char *sia_pp;              /* for passport prototyping
    int *mech[SIASWMAX];       /* pointers to mech-specific data
                               /* allocated by mechanisms indexed
                               /* by the mechind argument
} SIAENTITY;
```

---

## 6.5 SIA パラメータの収集

SIA にはパラメータ収集コールバック機能があるため、どのようなグラフィカル・ユーザ・インタフェース (GUI) でもコールバックを利用することができます。`sia_collect_trm()` ルーチンは、端末パラメータの収集に使いま



す。sia\_\*() ルーチン呼び出すコマンドは、引数として、適切な収集ルーチンへのポインタを渡します。このため、セキュリティ・メカニズムでは、ユーザに入力を求めることができます。収集ルーチンの引数が NULL の場合、セキュリティ・メカニズムは、収集が許可されていないため、要求を満たすために他の引数を使用する必要があるものと判断します。NULL は、非対話式のコマンドの場合に使用されます。信頼性を向上するには、可能な限り収集ルーチンを使ってください。

can\_collect\_input 引数は、セッション処理に含まれており、警告やエラー・メッセージを出力できる状態のまま、入力用の収集機能を使用不能にします。収集ルーチンは、簡単なフォームとメニューによるデータ収集をサポートしています。ある程度のフィールド検証機能がサポートされており、パラメータの長さや内容 (英数字、数字のみ、英字のみ、および非表示) がチェックされます。セキュリティを扱うコマンドやユーティリティで用意される収集ルーチンでは、表示特性を適切に設定する必要があります。

パラメータの収集機能は、sia.h で定義されている sia\_collect\_trm() インタフェースを使用する SIA によって提供されます。例 6-2 を参照してください。

#### 例 6-2: sia.h でのパラメータ収集インタフェースの定義

```
int sia_collect_trm(timeout, rendition, title,
                    num_prompts, prompts);

int timeout                /* number of seconds to wait */
/* 0 => wait forever */
int rendition
SIAMENUONE      1        /* select one of the choices given */
SIAMENUANY      2        /* select any of the choices given */
SIAFORM         3        /* fill out the form */
SIAONELINER     4        /* One question with one answer */
SIAINFO         5        /* Information only */
SIAWARNING      6        /* ERROR or WARNING message */

unsigned char *title       /* pointer to a title string. */ /*
/* NULL => no title */

int num_prompts           /* Number of prompts in collection */
prompt_t *prompts        /* pointer to prompts */

typedef struct prompt_t
{
    unsigned char *prompt;
```

## 例 6-2: sia.h でのパラメータ収集インタフェースの定義 (続き)

---

```
unsigned char *result;
int max_result_length; /* in chars */
int min_result_length; /* in chars */
int control_flags;
} prompt_t;

control_flags
SIARESINVIS 0x2  result is invisible
SIARESANY   0x10 result can contain any ASCII chars
SIAPRINTABLE 0x20 result can contain only printable chars
SIAALPHA     0x40 result can contain only letters
SIANUMBER    0x80 result can contain only numbers
SIAALPHANUM  0x100 result can contain only letters and numbers
```

---

パラメータ収集についての詳細は、`sia_collect_trm(3)` のリファレンス・ページを参照してください。

## 6.6 状態の保守

一部のコマンドでは、`sia_*`() への呼び出しを複数回行うため、これらの複数の呼び出しに渡って状態を保守する必要があります。状態は必ず、特定のユーザ (エンティティとも呼ばれる) に対応しています。SIA ではエンティティという用語を、認証できるユーザ、プログラム、またはシステムという意味で使います。エンティティ識別子は、ユーザ ID (UID) です。Tru64 UNIX に移植されたすべてのセキュリティ・メカニズムは、この UID が各メカニズムに同等にマッピングされるように管理されなければなりません。この条件により、複数のセキュリティ・メカニズムを共存させ、連携して動作させることができます。セキュリティ・メカニズムにユーザの代替識別子がある場合は、他のメカニズムが正しく連携して動作するようにし、同期のとれたセキュリティ情報を提供するために、一意の UID へのマッピングを行わなければなりません。

SIAENTITY 構造体 (6.4 節を参照) へのポインタは、セキュリティ・セッション機能を要求しているエンティティを識別するための中間状態を持つ引数として使います。また、SIAENTITY 構造体を使うと、セッション処理中にセキュリティ・メカニズム間で状態を共有することができます。

libc ライブラリは、SIAENTITY 構造体の基本要素の割り当てと解放を行う機能を提供しています。SIAENTITY 構造体の割り当ては、セッション初期化ルーチン `sia_ses_init()` の一部として行われます。SIAENTITY 構造体の割り当て解除は、セッション解放ルーチン `sia_ses_release()` への呼び出しで行われます。セッション処理中 (たとえば、`sia_ses_*authent()` ルーチンの中) にエラーが発生し、再試行せずにあきらめるときには、`sia_ses_release()` を呼び出して、そのセッションに関する SIAENTITY 構造体をクリーンにする (つまり解放する) 必要があります。`sia_ses_estab()` ルーチンまたは `sia_ses_launch()` ルーチンの途中でエラーが発生して障害状態が戻された場合は、これらのルーチンが `sia_ses_release()` を呼び出します。

## 6.7 SIA のリターン値

SIA は、成功または失敗を示す応答を、呼び出し元のコマンドまたはユーティリティへ戻します。SIAENTITY 構造体には、エラー・コード・フィールド (`error`) があり、エラーの詳細の定義に利用できます。

`siad_ses_*()` ルーチンは、次の状態を示すビットマップ値を戻します。

SIADFAIL	条件付きの失敗を示します。最下位ビットに 0 が設定されます。次のセキュリティ・メカニズムを引き続き呼び出します。
SIADSUCCESS	条件付きの成功を示します。最下位ビットに 1 が設定されます。
SIADSTOP	「無条件」の戻りを示します。最下位から 2 番目のビットに 1 が設定されます。SIADFAIL または SIADSUCCESS のどちらかと一緒に設定されます。

## 6.8 SIA のデバッグとログ

SIA は、デバッグおよびロギングの機能をサポートしており、データを `/var/adm/sialog` ファイルに追加することができます。SIA のロギング機能では、次の 3 つのログ項目タイプをサポートしています。

EVENT	SIA 処理中の成功の場合
-------	---------------

ERROR

SIA 処理中の失敗の場合

ALERT

SIA インタフェース内のセキュリティ構成  
またはセキュリティ・リスクの場合

sia\_log() のロギング・ルーチンは、セキュリティ・メカニズムで使用でき、printf() の書式と互換性のある書式付き文字列を受け入れます。各ログ・エントリにはタイムスタンプが付きます。例 6-3 は、一般的な /var/adm/sialog ファイルです。

#### 例 6-3: 一般的な /var/adm/sialog ファイル

---

```
SIA:EVENT Wed Feb  3 05:21:31 1999
Successful SIA initialization
SIA:EVENT Wed Feb  3 05:22:08 1999
Successful session authentication for terry on :0
SIA:EVENT Wed Feb  3 05:22:08 1999
Successful establishment of session
SIA:ERROR Wed Feb  3 05:22:47 1999
Failure to authenticate session for root on :0
SIA:ERROR Wed Feb  3 05:22:52 1999
Failure to authenticate session for root on :0
SIA:EVENT Wed Feb  3 05:22:59 1999
Successful session authentication for root on :0
SIA:EVENT Wed Feb  3 05:22:59 1999
Successful establishment of session
SIA:EVENT Wed Feb  3 05:23:00 1999
Successful launching of session
SIA:EVENT Wed Feb  3 05:24:40 1999
Successful authentication for su from root to terry
SIA:EVENT Wed Feb  3 05:25:46 1999
Successful password change for terry
```

---

sia\_log() ルーチンは、デバッグ用にのみ使用できます。\_ses\_\* ルーチンは、監査のログをとるために audgen() を使います。

## 6.9 SIA のセキュリティ・メカニズムの統合

要求されている SIA 処理のクラスまたはタイプによって、使用するセキュリティ・メカニズムとその順序は変化します。セキュリティ・メカニズムの組み合わせとしては、ローカル・メカニズム (ローカル・システム・セキュリティのみを扱う) と分散セキュリティ・メカニズム (いくつかのシステムに

またがったセキュリティ事項のみを扱う) が含まれるのが一般的です。SIA の階層構造によって、これら 2 つのセキュリティ・メカニズムを共存させたり、さらに進めて統合することができます。

セキュリティ・メカニズムの統合の例としては、ログイン処理やセッション処理があります。SIA の階層構造によって、セッション処理時に、各種のセキュリティ・メカニズム間で状態 (SIAENTITY) が受け渡されます。この状態には、収集された名前およびパスワードと、セッション処理の現状態が含まれます。ローカル・セキュリティ・メカニズムは、先に実行されたセキュリティ・メカニズムの認証処理を信頼するように設計することができます。これにより、認証の保証が可能になります。この場合、分散メカニズムでユーザが正常に認証されると、ローカル・メカニズムではその認証を信頼して受け入れ、セッション処理を継続することができます。

また SIA では、ローカル・メカニズムが保証を受け入れないようにすることもできます。この場合、以前の認証結果に関係なく、ローカル・メカニズムで独自の認証処理が行われます。その結果、ユーザはユーザ名とパスワードの入力を何度も求められることになります。SIA では任意の順序でセキュリティ・メカニズムを使用できますが、保証を受け入れるメカニズムを、保証を受け入れないメカニズムの後に使用する必要があります。

---

#### 注意

---

省略時のセキュリティ・メカニズム BASE は、認証の保証を受け入れます。

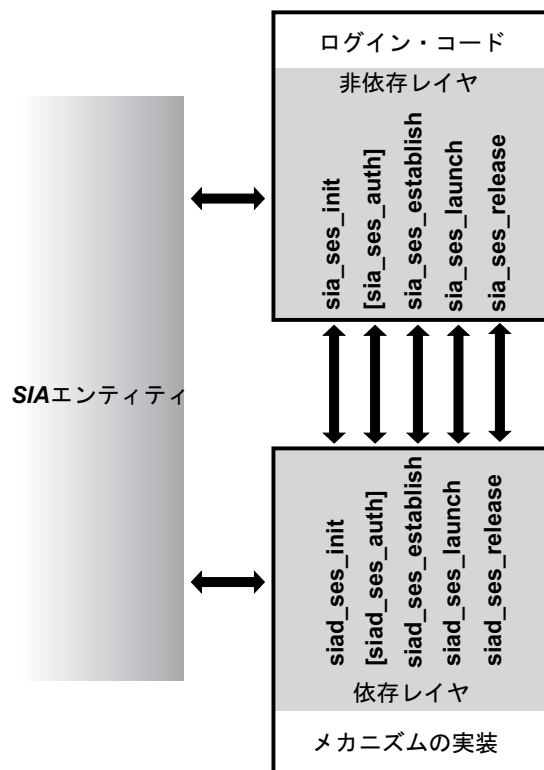
---

SIA のレイヤは、セキュリティ・メカニズムを、コマンドの個々のユーザ・インタフェースから分離する働きをします。この分離を行うために、呼び出し元のコマンドは、パラメータ収集ルーチンへのポインタを `sia_*()` への引数として渡します。収集ルーチンでは、簡単なフォームと、メニュー・タイプの処理をサポートしなければなりません。収集ルーチンの定義や必要条件は、`sia.h` に定義されています。セキュリティ・メカニズムからユーザ・インタフェースを分離することによって、ユーザ・インタフェースを柔軟に変更することができます。たとえば、ワークステーションに適應させたり、ダム端末モデルに適應させることができます。

## 6.10 SIA セッション処理

セッション処理インタフェースは、後で別のエンティティとなる必要がある、または別のエンティティとして動作する必要のある、ユーティリティまたはコマンドの処理に関連しています。図 6-2 は、一般的なログイン・セッションでの SIA ルーチンと、それらのルーチンの関係を示しています。

図 6-2: SIA セッション処理



ZK1085UAI

セキュリティ・メカニズムに依存するルーチン (`siad_*()`) へのセッション処理インタフェースでは、リターン値を使って、セッションの状態の判断と、継続するかどうかの判断を行います。リターン値は、次のとおりです。  
**SIADFAIL**

セキュリティ・メカニズムの `siad_*()` ルーチンからの **SIADFAIL** 応答は、セキュリティ・メカニズムが失敗したが、処理を継続する必要があることを示します。

## SIADFAIL | SIADSTOP

セキュリティ・メカニズムの `siad_*()` ルーチンからの SIADFAIL | SIADSTOP 応答は、セキュリティ・メカニズムが失敗し、セッション処理を停止する必要があることを示します。セキュリティ上の重大な問題またはリスクが見つかった場合に、この応答が使用されます。このような場合は、`siaolog` ファイルに ALERT イベントを送る必要があります。

## SIADSUCCESS

最後の応答は SIADSUCCESS で、セキュリティ・メカニズムがセッション処理のフェーズを正常に終了したことを示します。場合によっては、SIADSUCCESS | SIADSTOP というリターン値も使用できます。

すべてのセキュリティ・メカニズムが、セッション処理の各フェーズで必要な処理を備えているわけではありません。通常、省略時の応答は SIADFAIL です。この応答が戻されたときには、構成されている他のセキュリティ・メカニズムから、SIADSUCCESS 応答を得る必要があります。この唯一の例外は、セッション処理の最初と最後の段階です。セキュリティ・メカニズムでは、セッションの初期化またはセッションの解放時に何も行う必要がない場合、SIADSUCCESS 応答を返す必要があります。セッション処理のその他のフェーズでは、SIADFAIL 応答が省略時の値になります。

セッション処理インタフェースは通常、次の順序で呼び出されます。

<code>sia_ses_init()</code>	セッションを初期化します。
<code>sia_ses_authent()</code>	セッションの認証を行います。失敗したときには、再度呼び出して再試行することができます。
<code>sia_ses_estab()</code>	セッションを確立します。失敗した場合、 <code>sia_ses_release()</code> を呼び出します。
<code>sia_ses_launch()</code>	セッションを起動します。失敗した場合、 <code>sia_ses_release()</code> を呼び出します。
<code>sia_ses_release()</code>	セッションを解放します。

メカニズム・インデックス (mechind) の一貫性を保つために、すべてのセッション・ルーチンには、同じ数のメカニズムを同じ順序で指定する必要があります。

例 6-4 は、login コマンドのセッション処理を示すコードです。

#### 例 6-4: login コマンドのセッション処理

---

```
.
.
.
/* SIA LOGIN PROCESS BEGINS */

/* Logging of failures to sia_log is done within the libsia */
/* Logging to syslog is responsibility of calling routine */

if((sia_ses_init(&entity, oargc, oargv, hostname, loginname, \
    ttyn, 1, NULL)) == SIASUCCESS) {

    /***** SIA SESSION AUTHENTICATION *****/

    if(!fflag) {
        for(cnt=5; cnt; cnt--) {
            if((authret=sia_ses_authent(sia_collect,NULL,entity)) \
                == SIASUCCESS)
                break;
            else if(authret & SIASTOP)
                break;
            fputs(MSGSTR(INCORRECT, "Login incorrect\n"), stderr);
        }
        if(cnt <= 0 || (authret & SIASTOP)) {
            sia_ses_release(&entity);
            exit(1);
        }
    }

    /***** SIA SESSION ESTABLISHMENT *****/

    if(sia_ses_estab(sia_collect,entity) == SIASUCCESS) {
        /***** set up environment *****/
        /* destroy environ. unless user requested preservation */
        if (!pflag) {
            pp = getenv("TERM");
            if (pp)
                strncpy(term, pp, sizeof term);
            clearenv();
        }
        (void)setenv("HOME", entity->pwd->pw_dir, 1);
        if(entity->pwd->pw_shell && *entity->pwd->pw_shell)
```



#### 例 6-4: login コマンドのセッション処理 (続き)

```
        strncpy(shell, entity->pwd->pw_shell, sizeof shell);
(void)setenv("SHELL", shell, 1);
if (term[0] == ' ')
    (void)strncpy(term, stypeof(tty), sizeof(term));
(void)setenv("TERM", term, 0);
(void)setenv("USER", entity->pwd->pw_name, 1);
(void)setenv("LOGNAME", entity->pwd->pw_name, 1);
(void)setenv("PATH", _PATH_DEFPATH, 0);

/***** SIA LAUNCHING SESSION *****/

if(sia_ses_launch(sia_collect,entity) == SIASUCCESS) {
/* 004 - start */
if ((entity -> pwd          != NULL) &&
    (entity -> pwd -> pw_dir != NULL) &&
    (entity -> pwd -> pw_dir [0] != 0))
    sprintf (hush_path, "%s/%s",
            entity -> pwd -> pw_dir,
            _PATH_HUSHLOGIN);
else    strcpy (hush_path, _PATH_HUSHLOGIN);
quietlog = access(hush_path, F_OK) == 0;
/* 004 - end */
if(!quietlog)
    quietlog = !*entity->pwd->pw_passwd && \
!usershell(entity->pwd->pw_shell);
    if (!quietlog) {
        struct stat st;
        motd();
        (void)sprintf(tbuf, "%s/%s", _PATH_MAILDIR, \
entity->pwd->pw_name);
        if (stat(tbuf, &st) == 0 && st.st_size != 0)
            (void)printf(MSGSTR(MAIL, "You have %smail.\n"),
                (st.st_mtime > st.st_atime) ? MSGSTR(NEW, \
"new ") : );
    }
    sia_ses_release(&entity);

/***** Setup default signals *****/

(void)signal(SIGALRM, SIG_DFL);
(void)signal(SIGQUIT, SIG_DFL);
(void)signal(SIGINT, SIG_DFL);
(void)signal(SIGTSTP, SIG_IGN);

tbuf[0] = '-';
(void)strcpy(tbuf + 1, (p = rindex(shell, '/')) ?
    p + 1 : shell);
```

#### 例 6-4: login コマンドのセッション処理 (続き)

---

```
/****** Nothing left to fail *****/

    if(setreuid(geteuid(),geteuid()) < 0) {
        perror("setreuid()");
        exit(3);
    }
    execlp(shell, tbuf, 0);
    (void)fprintf(stderr, MSGSTR(NO_SHELL, \
"login: no shell: %s.\n"), strerror(errno));
    exit(0);
}
/****** SIA session launch failure *****/
}
/****** SIA session establishment failure *****/
}
    logerror(entity);
    exit(1);
}

logerror(entity)
SIAENTITY *entity;
{
    if(entity != NULL)
    {
        sia_ses_release(&entity);
    }
    syslog(LOG_ERR, MSGSTR(FAILURE3, " LOGIN FAILURE "));
}
.
.
.
```

---

#### 6.10.1 セッションの初期化

セッションの初期化は、`sia_ses_init()` ルーチンで行います。  
`sia_ses_init()` ルーチンは、構成されている各セキュリティ・メカニズムの `siad_ses_init()` エントリ・ポイントを呼び出し、セッション処理の開始に関連する処理を行います。セッションの初期化段階では、`SIAENTITY` 構造体を初期化する必要があります。この構造体は、セッション処理のさまざまな段階にわたって状態を管理するために使います。

## 6.10.2 セッションの認証

セッション処理の認証段階では、セッションの ID を割り当てます。セッション処理のこの段階では、セッションに対応させるエンティティを決定しなければなりません。エンティティを決定できない場合、認証は失敗します。認証に成功したら、エンティティが得られます。

最上位の SIA セッション認証ルーチン `sia_ses_authent()` は、`matrix.conf` ファイルに格納されている構成済の順序に従って、セキュリティ・メカニズムに依存する `siad_ses_authent()` を呼び出します。複数の認証ルーチンが呼び出されるため、SIAENTITY 構造体を使って、事前収集パラメータ (名前、パスワード、および最終的にはエンティティに対応する `/etc/passwd` エントリなど) が保持されます。

事前収集パラメータを使うことによって、セキュリティ・メカニズムは引数を再収集する必要がなくなります。たとえば、最初に DCE の `siad_ses_authent()` ルーチンを呼び出し、次にローカルの ENHANCED (エンハンスド・セキュリティ) の `siad_ses_authent()` を呼び出すように構成されているシステムに `root` がログインしようとする場合があります。

通常、DCE の認証処理では、`root` を認証することはできません。ただし、ユーザに名前とパスワードを求めることはできます。これらの情報は、SIAENTITY 構造体を使って ENHANCED の `siad_ses_authent()` ルーチンに渡されます。これにより、ENHANCED メカニズムで `root` の名前とパスワードを確認することができ、`root` を認証することができます。セッションの認証段階が終了するとすぐに、パスワード・フィールドがクリアされます。

セキュリティ・メカニズムに依存する認証ルーチンは、認証に成功したときにエンティティを決定して設定する機能を持たなければなりません。セキュリティ・メカニズムで、エンティティを独自に解釈する場合、共通の SIA エンティティ (ユーザ名と UID) への変換機能を提供しなければなりません。この制約なしでは、セキュリティ・メカニズム間でのエンティティの一貫性を確保することができません。

セッションの認証段階が正常に終了したときには、SIAENTITY 構造体に、認証されたエンティティのユーザ名とユーザ ID が格納されていなければなりません。セッションの認証に失敗した場合、呼び出し元のコマンドまたはプログラムは、`sia_ses_authent()` を再度呼び出して認証処理

を再試行することができます。一部のメカニズムでは、セッション処理のこの段階を、別のメカニズムに代行させることがあります。このような状況が発生するのは、ローカル・メカニズムが認証処理を省略し、別の分散メカニズムにゆだねた場合です。

### 6.10.3 セッションの確立

セッションの確立段階は、セッションの認証段階が正常に終了した後に、`sia_ses_estab()` で起動されます。`sia_ses_estab()` ルーチンは、複数のセキュリティ・メカニズムの `siad_ses_estab()` ルーチンを、`matrix.conf` ファイルに定義されている順序で呼び出すように構成されています。セッション処理のセッション確立段階では、メカニズム・リソースとライセンスをチェックし、このセッションを正常に起動できるか確認します。`passwd struct` エントリとその他の必要なセキュリティ・コンテキストの確認は、この段階で行う必要があります。セッション確立段階が正常に終了すると、システムはセッションを起動できる状態になります。

### 6.10.4 セッションの起動

セッションの起動段階では、セッションのスタートアップに関するロギングとアカウント処理を行います。ローカル・メカニズムでは、さらに `wtmp` エントリおよび `utmp` エントリを設定し、エンティティに対応する UID を実効 UID として設定します。`lastlog` の更新の他に、`setgid()` および `initgroup()` ルーチンによる処理も、ローカル・メカニズムで行います。重大なエラーが検出された場合だけ、セッションを継続せずに停止します。

### 6.10.5 セッションの解放

セッション処理手順の最後の段階では、成功か失敗かにかかわらず、`sia_ses_release()` ルーチンを呼び出します。このルーチンは、`SIAENTITY` 構造体などの、セッション処理リソースすべてを解放します。構成されているメカニズムが呼び出され、セッションで不要となったリソースが解放されます。

### 6.10.6 特定のセッション処理

以降の項では、`login`、`rshd`、および `rlogind` コマンドに特有のセッション処理について説明しています。

#### 6.10.6.1 login プロセス

最も一般的なセッション処理は、login プロセスが、ユーザに対応するエンティティになるときです。エンティティは、認証を行って権限が与えられる人またはプロセスの一意の SIA 識別子です。例 6-4 のコードは、login コマンドの一部分です。

#### 6.10.6.2 rshd プロセス

/usr/sbin/rshd のセッション処理は、login とは異なります。rshd プロセスは、ruserok() を呼び出し、.rhosts ファイルと host.equiv ファイルを参照して承認処理を行います。ruserok() が失敗すると、rshd も失敗します。

#### 6.10.6.3 rlogind プロセス

rlogind プログラムは、ruserok() の呼び出しが成功した場合、-f フラグを付けて login コマンドを実行します。ruserok() の呼び出しが失敗した場合には、-f フラグを付けずに実行します。-f フラグを付けずに login を実行した場合は、ユーザ名とパスワードの入力を求める sia\_ses\_authent() が必要に応じて呼び出されます。

### 6.11 セキュア情報の変更

この節で説明しているルーチンは、従来の /etc/passwd エントリ情報の変更を行います。このクラスのルーチンは、その他のタイプの共通セキュア情報も取り扱うように拡張することができます。従来の passwd, chfn, および chsh タイプのコマンド処理だけが規定されています。これらの各ルーチンは、同じオペレーション・モデルに従っています。ユーザが変更を要求すると、このクラスのルーチンは、構成された各メカニズムの siad\_chk\_user() を呼び出して、ユーザがそのメカニズムに登録されているか調べます。ユーザが 2 つ以上のセキュリティ・メカニズムに登録されている場合は、変更対象のメカニズムを選択できるように、収集ルーチンが選択メニューを表示します。要求を処理するメカニズムが 1 つだけ構成されている場合は、そのメカニズムが直接呼び出されます。

#### 6.11.1 ユーザのパスワードの変更

パスワードを変更するために、sia\_chg\_password() は siad\_chg\_password() ルーチンを使って、構成されているメカニズムを呼

び出します。どのメカニズムがそのユーザをサポートしているかを調べるために、`siad_chg_passwd()` ルーチンが構成されているすべてのメカニズムに対して、`siad_chk_user()` が呼び出されます。ユーザが複数のメカニズムに登録されている場合、そのユーザには選択肢が表示されます。ユーザが1つのメカニズムだけに登録されている場合は、そのメカニズムが呼び出されます。

### 6.11.2 ユーザの `finger` 情報の変更

`sia_chg_finger()` ルーチンは、構成されているメカニズムを `siad_chg_finger()` ルーチンを使って呼び出し、`finger` 情報を変更します。どのメカニズムがそのユーザをサポートしているかを調べるために、`siad_chg_finger()` ルーチンが構成されているすべてのメカニズムに対して、`siad_chk_user()` が呼び出されます。ユーザが複数のメカニズムに登録されている場合、そのユーザには選択肢が表示されます。ユーザが1つのメカニズムだけに登録されている場合は、そのメカニズムが呼び出されます。

### 6.11.3 ユーザのシェルの変更

`sia_chg_shell()` ルーチンは、構成されているメカニズムを `siad_chg_shell()` ルーチンを使って呼び出し、ユーザのログイン・シェルを変更します。どのメカニズムがそのユーザをサポートしているかを調べるために、`siad_chg_shell()` ルーチンが構成されているすべてのメカニズムに対して、`siad_chk_user()` が呼び出されます。ユーザが複数のメカニズムに登録されている場合、そのユーザには選択肢が表示されます。ユーザが1つのメカニズムだけに登録されている場合は、そのメカニズムが呼び出されます。

## 6.12 セキュリティ情報へのアクセス

以降の項で説明している SIA インタフェースは、従来の UNIX の `/etc/passwd` 情報および `/etc/group` 情報へのアクセスを処理します。他の共通のセキュア情報へのアクセスを処理するルーチンを作成することもできます。メカニズムに依存するセキュリティ情報へのアクセスは、アクセス対象の情報タイプを大半のメカニズムがサポートしている場合以外は、SIA インタフェースでは処理しないでください。

sia.h に定義されている `sia_context` 構造体および `mech_contexts` 構造体は、メカニズム間で状態を受け渡すために使います。これらの構造体を、次に示します。

```
struct mech_contexts {
    void *value;
    void (*destructor)();
};

struct sia_context {
    FILE *fp;
    union {
        struct group *group;
        struct passwd *pass;
    } value;
    int pkgind;
    unsigned buflen;
    char *buffer;
    struct mech_contexts mech_contexts[SIASWMAX];
};
```

`getgr*()` および `getpw*()` ルーチンには SIA インタフェースがあるため、セキュリティ・メカニズムでは、再入可能および再入不可の両方のスレッド・セーフ・アプリケーション用のルーチンを 1 つだけ用意する必要があります。この機能は、引数をセキュリティ・メカニズムの `siad_*()` ルーチン用の共通形式にカプセル化する、`sia_getpasswd()` および `sia_getgroup()` ルーチンによって実現されます。

### 6.12.1 /etc/passwd 情報へのアクセス

従来の `/etc/passwd` エントリは、`libc` および `libc_r` 内の `getpw*()` ルーチンを使ってアクセスされます。SIA レイヤの `sia_getpasswd()` ルーチンは、呼び出し方法が現在の `getpw*()` ルーチンと同じですが、この呼び出し方法を、シングルスレッドおよびマルチスレッドの両方のプロセスで利用できる、1 つの共通ルーチンに変換します。この変換を行うことで、セキュリティ・メカニズムでは 1 組の `getpw*()` ルーチンをサポートするだけで済みます。`getpwent()` ルーチンの処理は、すべてのエントリが処理されるまで、構成されている各セキュリティ・メカニズムを、事前に定義された順序で呼び出すことで行われます。

### 6.12.2 /etc/group 情報へのアクセス

従来の /etc/group エントリは、libc および libc\_r 内の `getgr*()` ルーチンを使ってアクセスされます。SIA レイヤの `sia_getgroup()` ルーチンは、呼び出し方法が現在の `getgr*()` ルーチンと同じですが、この呼び出し方法を、シングルスレッドおよびマルチスレッドの両方のプロセスで使用できる、1つの共通ルーチンに変換します。1つのルーチンに変換することにより、必要なルーチンの数が少なくなるため、セキュリティ・メカニズムの移植性が向上します。`getgrent()` ルーチンの処理は、すべてのエントリが処理されるまで、構成されている各セキュリティ・メカニズムを、事前に定義された順序で呼び出すことで行われます。

### 6.13 セッション・パラメータの収集

SIA セッション・インタフェースと、セキュア情報を変更するインタフェースでは、事前に定義されたパラメータ収集機能を使います。呼び出し元のアプリケーションは、パラメータ収集ルーチンのアドレスを、SIA を介して `siad_*()` ルーチンへ渡します。収集ルーチンは、ユーザ・インタフェースの詳細を意識することなく、さまざまなセキュリティ・メカニズムを使って、さまざまなパラメータの入力をユーザに求めることができます。

この機能により、SIA セキュリティ・メカニズムを、ユーザ・インタフェースや、簡単なフォームやメニューによる入力処理から独立させることができます。この収集機能には、各種のユーザ・インタフェース・パッケージやウィンドウ・システムで簡単に実装できるように、制限が設けられています。ただし、収集ルーチンは、簡単な (8 項目までの) メニューと、フォーム・スタイルの処理をサポートしていなければなりません。ダム端末では、フォーム処理は 1 行単位の質問のセットになります。この機能がないと、セキュリティ上の新しい質問をサポートするために、アプリケーションを変更する必要が生じます。

### 6.14 SIA 製品のパッケージ

SIA では、Tru64 UNIX システムへ移植するために必要なセキュリティ・メカニズム構成要素を定義しています。これらの構成要素は、次のとおりです。

- メカニズムに依存する (`siad_*()`) ルーチンを含むシェアード・ライブラリ。コマンドおよびユーティリティへのインタフェースとして使います。



- 省略時の構成ファイル `/etc/sia/matrix.conf`。このファイルは、SIA を介してセキュリティ・メカニズムを使うためにインストールされます。

シェアード・ライブラリには、表 6-3 で示すすべての `siad_*()` ルーチンが入っていない必要はありません。`siad_*()` ルーチン用の省略時のダミー・ルーチンは常に、失敗を示す `SIADFAIL` 応答を戻します。セキュリティ・メカニズムでダミー・ルーチンを用意している場合は、これらのルーチンを `matrix.conf` ファイルに構成しないでください。

`/etc/sia/matrix.conf` ファイルには、各 `siad_*()` ルーチンに対して行が 1 つあります。この行には、メカニズム識別子 (`mech_types` と呼ぶ) とセキュリティ・メカニズム・ライブラリへの実際のパスを記述します。`sia_*()` ルーチンは、この鍵のセットを使って、右から左の順番でメカニズムを呼び出します。Tru64 UNIX での `matrix.conf` の省略時の設定については、『セキュリティ管理ガイド』を参照してください。

DCE セキュリティ・メカニズムを最初に呼び出してから、BASE (BSD) セキュリティ・メカニズムを呼び出す場合は、`siad_init()` の構成行は次のようになります。

```
siad_init=(DCE,/usr/shlib/libdcesia.so) (BSD,libc.so)
```

レイヤード・セキュリティ・プロダクトでは、事前にテストされた `matrix.conf` ファイルをキットに含めて配布しなければなりません。SIA の `matrix.conf` ファイルを変更したときには、リブートを行わなければなりません。システム管理者は、使用中の `matrix.conf` ファイルを手作業で編集してはなりません。

`matrix.conf` ファイルについての詳細は、『セキュリティ管理ガイド』を参照してください。

## 6.15 セキュリティ・メカニズムに依存するインタフェース

セキュリティ・メカニズムには、すべての `siad_*()` エントリ・ポイントを用意する必要があります。表 6-3 を参照してください。省略時のスタブ・ルーチンでは、`SIADFAIL` を返す必要があります。セッション・ルーチンの場合を除き、`/etc/sia/matrix.conf` ファイルではスタブを呼び出してはなりません。メカニズム・インデックス (`mechind`) の一貫性を確保するために、すべてのセッション・ルーチンには、同じ順序で同じ数のメカニズムを記述する必要があります。ただし、構成に誤りがあった場合は、適切な `SIADFAIL` 応答をスタブ・ルーチンが戻します。

/etc/sia/matrix.conf ファイル内のセキュリティ・メカニズムの順序は、どのクラスのインタフェースでも同じです。このため、セキュリティ・メカニズムがセッション処理をサポートしている場合は、セッション関連のすべてのインタフェースに対して、セキュリティ・メカニズムが同じ順序で呼び出されます。

レイヤード・セキュリティ・メカニズムでは、プレフィックスが `mechanism_name__` のプライベート・エントリ・ポイント群を、各 `siad_*()` エントリに対して用意する必要があります。これらのエントリは、メカニズム内で `siad_*()` ルーチンを内部的に呼び出すために使用されます。この例は、libc 内の BASE メカニズム内にあります。BASE メカニズムが自身の `siad_getpwuid()` ルーチンを確実に呼び出すようにするには、別のエントリ・ポイントを作成し、`siad_getpwuid()` エントリから次のようにこのエントリ・ポイントを呼び出します。

```
int siad_getpwuid(uid_t uid, struct passwd *result, \
                  char *buffer, int buflen)
{
    return(bsd_siad_getpwuid(uid,result,buffer,buflen));
}

static int bsd_siad_getpwuid(uid_t uid, struct passwd *result, \
                             char *buffer, int buflen)
{
    /* The BSD security mechanism siad_getpwuid() routine */
}
```

この内部名が `siad_*()` エントリ・ポイントすべてに対して用意されている場合、レイヤード・セキュリティ・メカニズムは、セキュリティ・メカニズムに依存するコードすべてを別のライブラリとして作成することができます。これにより、構成されるシェアード・ライブラリは、別のライブラリを呼び出すスタブだけになります。

セキュリティ・メカニズムは一般に、ローカル・メカニズムと分散メカニズムの2つのカテゴリに分けられます。ローカル・セキュリティ・メカニズムは、ローカル・システム上のセッションを確立するために必要なローカル・コンテキストすべてを確立する責任があります。Tru64 UNIX には、BASE メカニズムと ENHANCED メカニズムの2つのローカル・セキュリティ・メカニズムがあります。

DCE などの分散メカニズムは、Kerberos チケットなどの分散セッション・コンテキストの確立に深く関わります。ただし、分散セキュリティ・メカニ

ズムも、ローカル・セキュリティ・メカニズムで使うことのできる、一部のローカル・コンテキストを提供することがあります。分散セキュリティ・メカニズムは、十分に強力な認証を提供することができるため、ローカル・メカニズムはこの認証に頼ることもできます。あるメカニズムが別のメカニズムを信頼するというこの概念は、保証と呼ばれます。この概念により、ログイン・セッションを確立するためのユーザの認証が、1回だけで済みます。ローカル・メカニズムは必ず、呼び出し順序の最後に構成する必要があります。

この節で示した SIA 機能はすべて、複数のセキュリティ・メカニズムを使うように構成できます。

## 6.16 シングルユーザ・モード

独自のシングルユーザ・セキュリティ・モードが必要な場合は、影響を受けるコマンドおよびユーティリティ (たとえば、`/sbin` にある、静的にリンクされたバイナリ) を再構築して置き換える必要があります。再構築および置き換えを行うには、影響を受けるコマンドのリンク順序として、`siad_*()` ルーチンのライブラリを指定してから `libc` を指定します。

新しいルーチンでは、`siad_*()` ルーチンではなく、`__siad_*()` ルーチンを置き換える必要があります。`siad_*()` 形式の名前は弱いシンボルのエントリ・ポイントですが、`__siad_*()` 形式の名前は実際に使用される強いシンボルのエントリ・ポイントです。ルーチンの命名規則についての詳細は、6.17 節を参照してください。

## 6.17 SIA ルーチンのシンボル優先使用

この節では、開発者が追加するルーチンを ANSI C ルーチンの命名規則に準拠させるためのルーチン命名規則について説明しています。

### 6.17.1 シンボル優先使用の問題の概要

`libc.a` の前にロードされるライブラリに SIA ルーチンと同じ名前のルーチン (たとえば、`siad_ses_init()`) を用意するだけでは、`libc` 内の SIA ルーチンが使用しているシンボルを無効にすることはできません。これは、ANSI C での `libc` のルーチン名の命名規則と、ユーザ用に予約されたシンボルのためです。

`libc.a` ライブラリと `libc.so` ライブラリ内にどのようなエントリ・ポイントを置けるかについて、ANSI C の必要条件とアプリケーション開発者の要望

に食い違いがあります。ANSI C 標準では、使用可能なシンボルをリストしています。そして、これらのシンボル以外を置くときには、「ベンダ用に予約された」形式でなければなりません。つまり、これらのシンボルは2つの下線で始まるか、1つの下線と大文字で始まらなければなりません。これらのシンボルは制限されているので、一般的なユーザの要望には合っていないです。

### 6.17.2 Tru64 UNIX での解決方法

ANSI C と開発者の要望の両方を満たすために、Tru64 UNIX では、「強い」シンボルおよび「弱い」シンボルを使って追加の名前を提供しています。たとえば、`bcopy()` というルーチンが ANSI C で使用できない場合は、`bcopy()` という名前の弱いシンボルと `__bcopy()` という名前の強いシンボルを用意します。

弱いシンボルは、`libc` 内の `bcopy()` ルーチンに影響することなく、ユーザが優先使用することができます。ライブラリでは、これらの「名前スペース保護」ルーチンに対して、強いシンボルを使うためです。

SIA ルーチンの場合、`siad_ses_init` という名前の弱いシンボルがあり、通常は強いシンボル `__siad_ses_init()` にバインドされることを意味します。別のコードが既にシンボル `siad_ses_init()` を使っている場合、弱いシンボルのバインドだけが影響を受けます。

`libc` 内の SIA コードは、自分自身が使用する場合には、強いシンボル `__siad_ses_init()` を参照します。このため、シングルユーザ・モードの省略時の BASE セキュリティ・メカニズムを無効にするには、`__siad_ses_init()` ルーチンを置き換えるルーチンを用意する必要があります。

SIA ルーチンと `/etc/sia/matrix.conf` ファイルの制御下で動的にのみロードされるライブラリの場合は、`siad_ses_init()` 形式のシンボル名だけを用意すれば済みます。動的にロードされるライブラリが必ず `matrix.conf` ファイルを介して使用される場合は、シンボルの両方の形式を提供することもできます。これにより、コードが簡略化されます。ただし、ライブラリの使い方が変わって、動的なロードだけではなくリンクが必要になった場合は、安全ではなくなります。

### 6.17.3 シングルユーザ環境の置き換え

例 6-5 は、セキュリティ・メカニズム・ライブラリの開発者が、通常のシェアード・ライブラリを `matrix.conf` 用に提供する他に、シングルユーザ環境を置き換える必要がある場合に使うコードを示しています。

#### 例 6-5: シングルユーザ環境のシンボル優先使用

---

```
/* preempt libc.a symbols in single-user mode */
#ifdef SINGLE_USER
# pragma weak siad_ses_init = __siad_ses_init
# define siad_ses_init __siad_ses_init
#endif
#include <sia.h>
#include <siad.h>
```

---

シングルユーザ (静的) ライブラリのモジュールは、その後、次のようにコンパイルされます。

```
% cc -DSINGLE_USER ...
```

これで、シェアード・ライブラリが `libc.so` のシンボルに干渉するのを防ぎますが、シングルユーザ・モードで使用する非共用イメージの `libc.a` のシンボルの優先使用を可能にします。そして、非共用イメージは次の例のように、リンカへのパラメータとして `libc.a` の前に置き換え版のメカニズム・ライブラリを指定して構築されます。

```
% cc -non_shared -o passwd passwd.o -ldemo_mech
```

シェアード・ライブラリは、通常の方法で構築されます。



## ACL のプログラミング

この章では、次の内容について説明しています。

- ACL の概要
- ACL のデータ表現
- ACL ライブラリ・ルーチン
- ACL の規則
- ACL の作成例
- ACL の継承例

### 7.1 ACL の概要

Tru64 UNIX のアクセス制御リスト (ACL) は、UNIX システムで以前から提供されていた任意アクセス制御 (DAC: Discretionary Access Control) に対する拡張です。この拡張はオプションです。従来の UNIX DAC は、従来の UNIX 許可ビットです。ACL は、UNIX 許可ビットの拡張です。許可ビットだけを持つファイルまたはディレクトリは、`usr`、`group`、および `other` の許可ビットに対応する 3 つの必須エントリ (つまり基本エントリ) だけからなる ACL を持つオブジェクトと考えられます。

ACL には次の 2 種類があります。

- アクセス **ACL** は、ファイルまたはディレクトリと対応していて、プロセスがそのファイルやディレクトリにアクセスできるか判断するために使われます。
- 省略時の **ACL** は、ディレクトリと対応しています。省略時の ACL は、そのディレクトリに作成される新しいファイルおよびサブディレクトリに適用する ACL を決定するために使われます。詳細については、7.6 節を参照してください。

Tru64 UNIX での ACL の実装は、POSIX P1003.6 標準の Draft 13 と、Draft 15 の拡張の一部に基づいています。

ACL は、プロパティ・リストをサポートしているファイル・システムのファイルやディレクトリに適用することができます。プロパティ・リストをサポートしているファイル・システムは、次のとおりです。

- UFS
- AdvFS
- NFS (Tru64 UNIX システム間)

ACL の処理がシステムで有効になっていなくても、ACL を適用することができます。ただし、ACL のアクセス・チェックおよび省略時の ACL 継承は実行されません。

ACL の使用および管理についての詳細は、『セキュリティ管理ガイド』を参照してください。ACL の使用およびプログラミングについての説明は、リファレンス・ページの `acl(4)` を参照してください。プロパティ・リストについての詳細は、リファレンス・ページの `proplist(4)` を参照してください。

## 7.2 ACL のデータ表現

ACL には、内部表現と外部表現があります。外部表現は、テキスト形式であり、ACL の入力および表示に使います。ライブラリ・ルーチンは、作業域内の ACL を内部表現で処理します。この ACL は、呼び出し元のルーチンでは間接的にしかアクセスできません。この内部表現は、ヘッダ・ファイル `acl.h` を使って処理することができます。

### 7.2.1 内部データ表現

ACL ルーチンは、作業域表現を処理します。この表現は、ACL および ACL エントリのデータ構造体のセットからなり、隠されています。ユーザのプログラムは、定義されたルーチンを介して、これらのデータ構造体を処理する必要があります。作業域のデータ構造体は変更される可能性があるため、このデータの信頼できるアクセス方法は、定義されたインタフェースを使用する方法だけです。

作業域表現は、メモリ内では連続していません。また、プログラムでは、ACL エントリおよび ACL 記述子のサイズを調べることはできません。作業域のデータ構造体には、内部ポインタ参照が含まれているため、プロセス間で引き渡したり、ファイルに格納すると意味がなくなります。プログラムでは、ACL の作業域表現を ACL の別の表現に変換することができます。



隠されているデータにアクセスするために最もよく使われる 2 つの型は，acl (ACL 構造体) 型へのポインタ `acl_t` と，ACL エントリ構造体へのポインタ `acl_entry_t` です。

---

注意

---

以降の項の構造体は，変更される可能性がある，不透過な内部データ構造体です。これらの構造体にアクセスするときには，定義されている型と，提供されているライブラリ・ルーチンを必ず使用してください。

---

内部表現では，以下の基本型とデータ構造体を使います。

#### 7.2.1.1 `typedef struct acl *acl_t;`

`acl_t` 型は，内部 (作業域) 形式の ACL を指定するために使います。

```
struct acl {
    int      acl_magic;      /* validation member */
    int      acl_num;        /* number of actual acl entries */
    int      acl_alloc_size; /* size available in the acl */
    acl_entry_t acl_current; /* pointer to current entry in */
    acl_entry_t acl_first;   /* pointer to ACL linked list */
    attribute_t *attr_data; /* Pointer to the attr data */
};
```

#### 7.2.1.2 `typedef struct acl_entry *acl_entry_t;`

`acl_entry_t` 型は，ACL 内のエントリを指定するために使います。

```
struct acl_entry{
    acl_t      *entry;
    void      *head;
    struct acl_entry *next;
    struct acl_entry *prev;
    int      acl_magic;
    int      size;
};
```

#### 7.2.1.3 `typedef uint_t acl_type_t;`

サポートされている ACL タイプは，次のとおりです。

```
#define ACL_TYPE_ACC 0
#define ACL_TYPE_ACCESS ACL_TYPE_ACC
/* The ACL is an access ACL. The property list
   entry name for an access ACL is "DEC_ACL_ACC" */
```

```

#define ACL_TYPE_DEF 1
#define ACL_TYPE_DEFAULT ACL_TYPE_DEF
/* The ACL is a default access ACL. The property list
   entry name for a default access ACL is "DEC_ACL_ACC" */
#define ACL_TYPE_DEF_DIR 2
#define ACL_TYPE_DEFAULT_DIR ACL_TYPE_DEF_DIR
/* The ACL is a default directory ACL. The property list
   entry name for a default directory ACL is "DEC_ACL_DEF_DIR" */

```

acl\_type\_t は、ACL のタイプを指定するために使います。

#### 7.2.1.4 typedef uint acl\_tag\_t;

acl\_tag\_t 型は、ACL エントリのタグ (タイプ) を指定するために使います。ACL\_USER または ACL\_GROUP のタグ・タイプを持つ ACL エントリには、対応するタグ修飾子もあります。タグ修飾子は、ユーザ ID または グループ ID です。サポートされる、ACL エントリのタグ・タイプは次のとおりです。

```

#define ACL_USER_OBJ 0
/* entry that equates to the owning user permission bits. */
#define ACL_GROUP_OBJ 1
/* entry that equates to the owning group permission bits. */
#define ACL_OTHER 2
#define ACL_OTHER_OBJ ACL_OTHER
/* entry that equates to the other permission bits. */
#define ACL_USER 23
/* entry specifying permissions for a given user. */
#define ACL_GROUP 24
/* entry specifying permissions for a given group. */

```

#### 7.2.1.5 typedef uint\_t acl\_perm\_t;

許可ビット acl\_perm\_t の定義は、次のとおりです。

```

#define ACL_EXECUTE 0X001
#define ACL_WRITE 0X002
#define ACL_READ 0X004

```

#### 7.2.1.6 typedef acl\_perm\_t \*acl\_permset\_t;

acl\_permset\_t 型は、ACL エントリに割り当てられた許可ビットへのポインタとして使います。

7.2.1.7 連続内部表現 ACL

連続し、持続性のある、ACL のデータ型もあります。この表現は、複数のプロセス間で内部形式 ACL を持続させる必要があるときだけ使用します。

7.2.2 外部表現

人が読むことのできる ACL 外部表現は、改行文字で終了する、複数の行の並びで構成されています。POSIX ルーチンは、作業域表現とテキスト・パッケージとの間で変換を行う際に外部表現を使います。

外部表現については、『セキュリティ管理ガイド』を参照してください。表 7-1 に、各エントリの構造を示します。

表 7-1: ACL エントリの外部表現

エントリ・タイプ	acl_tag_t の値	エントリ
基本ユーザ	USER_OBJ	user::許可
基本グループ	GROUP_OBJ	group::許可
基本その他	OTHER_OBJ	other::許可
ユーザ	USER	user:ユーザ名:許可
グループ	GROUP	group:グループ名:許可

7.3 ACL ライブラリ・ルーチン

ACL ルーチンは、libpACL.a ライブラリ内にあります。ACL ライブラリ・ルーチンは、POSIX P1003.6 標準の Draft 13 に基づいています。詳細については、それぞれのルーチンのリファレンス・ページを参照してください。

以下のルーチンは、ACL の取得、設定、および確認に使用します。

acl_valid()	指定された内部表現 ACL が、正しいフォーマットかどうかをチェックします。
acl_delete_def_fd()	ファイル記述子を使って、指定ディレクトリから省略時のアクセス ACL を削除します。

<code>acl_delete_def_file()</code>	指定ディレクトリから省略時のアクセス ACL を削除します。
<code>acl_get_fd()</code>	ファイル記述子を使って、特定のファイルまたはディレクトリに対応する、指定の ACL タイプの内部表現を取り出します。
<code>acl_get_file()</code>	特定のファイルまたはディレクトリに対応する、指定の ACL タイプの内部表現を取り出します。
<code>acl_set_fd()</code>	ファイル記述子を使って、特定のファイルまたはディレクトリ上の指定の ACL タイプに、指定の ACL 内部表現を設定します。
<code>acl_set_file()</code>	特定のファイルまたはディレクトリ上の指定の ACL タイプに、指定の ACL 内部表現を設定します。
以下のルーチンは、ACL エントリの取り出しや処理を行います。	
<code>acl_copy_entry()</code>	ACL エントリを、指定されたメモリにコピーします。
<code>acl_create_entry()</code>	必要に応じてメモリを割り当てながら、指定された ACL に対して空の ACL エントリを作成します。
<code>acl_delete_entry()</code>	指定された ACL エントリを、ACL から削除します。
<code>acl_first_entry()</code>	現在の ACL エントリをリセットし、 <code>acl_get_entry()</code> の次の呼び出しで最初のエントリが戻されるようにします。

<code>acl_get_entry()</code>	指定された ACL の、次の ACL エントリへのポインタを戻します。
------------------------------	-------------------------------------

以下のルーチンは、ACL エントリ内のフィールドの取り出しや、処理を行います。

<code>acl_add_perm()</code>	ACL エントリに指定されている許可一式に、許可を 1 つ追加します。
-----------------------------	-------------------------------------

<code>acl_clear_perm()</code>	指定された ACL エントリの許可をクリアします。
-------------------------------	---------------------------

<code>acl_delete_perm()</code>	ACL エントリに指定されている許可一式から、許可を削除します。
--------------------------------	----------------------------------

<code>acl_get_permset()</code>	指定された ACL エントリから、指定されたメモリ位置に許可をコピーします。
--------------------------------	--

<code>acl_get_qualifier()</code>	指定された ACL エントリに対応するタグ修飾子 (ID) へのポインタを戻します。
----------------------------------	--

<code>acl_get_tag_type()</code>	指定された ACL エントリから、指定されたメモリ位置にタグ (タイプ) をコピーします。
---------------------------------	---

<code>acl_set_permset()</code>	指定された ACL エントリ内の許可に、指定された許可を設定します。
--------------------------------	------------------------------------

<code>acl_set_qualifier()</code>	指定された ACL エントリのタグ修飾子 (ID) に、指定された UID または GID を設定します。
----------------------------------	---

<code>acl_set_tag_type()</code>	指定された ACL エントリのタグ (タイプ) に、指定されたタイプを設定します。
---------------------------------	---

以下のルーチンは、ACL 処理のための作業域を管理します。

<code>acl_free()</code>	指定された ACL に関連する作業域すべてを解放します。
<code>acl_free_qualifier()</code>	指定されたタグ修飾子に関連する作業域を解放します。
<code>acl_free_text()</code>	指定された外部表現 (テキスト) ACL に関連するバッファを解放します。
<code>acl_init()</code>	ACL の内部表現作業域を割り当て、初期化します。
<code>acl_copy_ext()</code>	作業域内部形式の ACL データを、連続し持続性のある ACL フォーマットにコピーします。
<code>acl_copy_int()</code>	連続し持続性のある ACL データを、作業域にコピーします。
<code>acl_dup()</code>	指定された ACL のコピーを作成します。コピーは、元のエントリとは独立したものになります。
<code>acl_size()</code>	指定された ACL のサイズを計算します。
以下のルーチンは、ACL の外部表現と内部表現の間の変換を行います。	
<code>acl_from_text()</code>	指定された外部表現 (テキスト) ACL から、内部表現 ACL を作成します。
<code>acl_to_text()</code>	指定された内部表現 ACL から、外部表現 (テキスト) ACL を作成します。

## 7.4 ACL の規則

ACL と、UNIX のアクセス許可の関係には、微妙な部分があります。  
ACL ルーチンと、UNIX DAC 属性を処理するシステム・コールの間の相

互作用を理解していないと、意図していないアクセス許可を設定してしまうことがあります。

以降の項では、ACL を処理するプログラムの規則について説明しています。

#### 7.4.1 オブジェクトの作成

ファイル・システム上で ACL が有効になっていて、サポートされている場合、関数 `open()`、`creat()`、および `mkdir()` は、ファイルまたはディレクトリを作成するときに ACL を継承させます。詳細は、ACL の継承の説明を参照してください。

ACL が継承されるとき、作成されたファイルのアクセス許可はユーザが指定したモードと継承された ACL から作成され、`umask` は使用されません。このため、ユーザのプログラムでは、ファイルおよびディレクトリの作成時にモードを設定する必要があります。プログラムでは、`umask` に依存してファイルおよびディレクトリを保護してはなりません。

プログラムで、あるファイルを別のファイルにコピーするときは、新しいファイルを作成して、所有者、グループ、およびモードを引き継がせるのが一般的です。コピー元ファイルに ACL がある場合、ユーザのプログラムでは、モードが引き継がれるすべてのケースで、ターゲット・ファイルに ACL を引き継がせる必要があります。

#### 7.4.2 ACL の複製

アクセス許可を複製するプログラムは、ACL をそのまま引き継がせなければなりません。ファイルまたはディレクトリの任意保護は、所有者、グループ、およびアクセス許可では表現されなくなります。この保護には、アクセス許可のスーパーセットである ACL が含まれています。ACL をコピーしないと、ファイルやディレクトリへの意図しないアクセスを許可してしまう可能性があります。

#### 7.4.3 ACL の有効性

ACL は、次の POSIX ACL 規則に従って、正しく作成しなければなりません。

- 少なくとも 3 つの基本エントリが含まれていること。
- ユーザ・エントリには、重複しない有効な修飾子があること。

- グループ・エントリには、重複しない有効な修飾子があること。
- ユーザおよびグループ識別子が有効であること。

`acl_valid()` ルーチンを使って、ACL をチェックすることができます。

## 7.5 ACL の作成例

次のアクセス許可のとおりに、ファイルのアクセス ACL を設定したいと思います。

```
user::rwx

user:june:r-x
user:sally:r-x
group::rwx
group:mtg:rwx
other::r-x
```

次のコードは、表形式の ACL を使って ACL の作業域表現を作成し、その表現をファイルに適用します。次のコードを `acl_example.c` という名前のファイルに抽出した場合は、次のコマンドを使ってコンパイルします。

```
# cc -o acl_example -lpacl -lsecurity acl_example.c

#include <unistd.h>
#include <sys/types.h>
#include <prot.h>
#include <errno.h>
#include <sys/acl.h>

struct entries {
    acl_tag_t      tag_type;
    char           *qualifier;
    acl_perm_t     perms;
} table[] = {
    /* An ACL must (at a minimum) have the three base */
    /* entries that correspond to the permission bits */
    { ACL_USER_OBJ,  NULL,      ACL_READ | ACL_WRITE | ACL_EXECUTE },
    { ACL_USER,     "june",    ACL_READ | ACL_EXECUTE },
    { ACL_USER,     "sally",   ACL_READ | ACL_EXECUTE },
    { ACL_GROUP_OBJ, NULL,      ACL_READ | ACL_WRITE | ACL_EXECUTE },
    { ACL_GROUP,    "mtg",     ACL_READ | ACL_WRITE | ACL_EXECUTE },
    { ACL_OTHER_OBJ, NULL,     ACL_READ | ACL_EXECUTE },
};

#define TABLE_ENTRIES (sizeof(table)/sizeof(table[0]))

main (argc, argv)
int argc;
```



```

char *argv[];
{
    acl_t acl_p;
    acl_entry_t entry_p;
    acl_entry_t check_p;
    int i, ret;
    uid_t uid;
    gid_t gid;

    /* Did the user enter a filename? */
    if (argc != 2) {
        printf("Usage: %s filename\n",argv[0]);
        exit(1);
    }

    /* Check to see if ACLs are supported and enabled for filename */
    ret = pathconf(argv[1], _PC_ACL_EXTENDED); /*1*/
    if (ret == 1) {
        printf(" ACLs are enabled for file %s\n",argv[1]);
    }
    else if (ret == 0) {
        printf(" ACLs are supported for file %s,\n",argv[1]);
        printf(" but ACLs are not currently enabled on \n");
        printf(" the system\n");
    }
    else if ((ret == -1) && (errno == EINVAL)) {
        printf(" ACLs not supported on filesystem\n");
        exit(1);
    }
    else {
        printf(" Error checking file ACL status for \n");
        printf(" file %s, exiting...\n",argv[1]);
        perror("pathconf");
        exit(1);
    }

    /* Allocate an ACL */
    acl_p = acl_init(1024); /*2*/

    /* Walk through the table creating corresponding ACL entries */
    for(i=0;i<TABLE_ENTRIES;i++) {

        /* Initialize the entry */
        entry_p = acl_create_entry(&acl_p); /*3*/

        /* Set the permissions for the entry */
        acl_set_permset(entry_p,&table[i].perms); /*4*/

        /* Set the user or group information for the entry */

```

```

switch(table[i].tag_type) {

    case ACL_USER:

        /* Get the uid from the user name */
        uid=pw_nametoid(table[i].qualifier);          /* 5 */
        if (uid == (uid_t) -1) {
            printf("  No translation for user name %s\n",
                table[i].qualifier);
            printf("  Exiting...\n");
        }
        exit(1);

        /* Specify this is a "USER:" entry */
        acl_set_tag_type(entry_p,table[i].tag_type);    /* 6 */

        /* Set the uid (entry qualifier) */
        acl_set_qualifier(entry_p,(void *)&uid);        /* 7 */
        break;

    case ACL_GROUP:

        /* Get the gid from the group name */
        gid=gr_nametoid(table[i].qualifier);          /* 8 */
        if (gid == (gid_t) -1) {
            printf("  No translation for group name %s\n",
                table[i].qualifier);
            printf("  Exiting...\n");
        }
        exit(1);

        /* Specify this is a "GROUP:" entry */
        acl_set_tag_type(entry_p,table[i].tag_type);

        /* Set the gid (entry qualifier) */
        acl_set_qualifier(entry_p,(void *) &gid);
        break;

    default:

        /* The three entries corresponding to the */
        /* Permission bits don't have qualifiers */
        acl_set_tag_type(entry_p,table[i].tag_type);
        acl_set_qualifier(entry_p,NULL);
        break;
}

/* Is the created ACL valid? */
if (acl_valid(acl_p, &check_p) < 0) {                /* 9 */

```

```

        printf(" Not Valid ACL\n");
        if (check_p) printf(" Duplicate entries\n");
        printf(" Exiting...\n");
exit(1);
    }

    /* Set the ACL on the file */
    if (acl_set_file(argv[1],ACL_TYPE_ACCESS, acl_p) < 0)
        perror("acl_set_file");

    /* Free the storage allocated for the ACL */
    acl_free(acl_p);
}

```

- ❶ pathconf() の `_PC_ACL_ENABLED` 属性には、指定されたファイルの ACL 処理のステータスが戻されます。
- ❷ これは、ACL の作業域表現の初期化呼び出しの使い方を示しています。割り当てられている領域に、完全な ACL のエントリすべてを収めるだけの大きさがなかった場合、`acl_create_entry()` ルーチンは追加のメモリを割り当てます。
- ❸ 新しい ACL エントリはこの呼び出しで割り当てられます。この新しいエントリのタグ・タイプ、修飾子、許可は指定されていません。
- ❹ `acl_set_permset()` ルーチンは、ACL エントリの許可を設定します。
- ❺ `pw_nametoid()` ルーチンは、ユーザ名からユーザ ID への最適化されたマッピングを提供し、Base セキュリティまたは Enhanced セキュリティのうち使用可能なほうを扱います。`pw_nametoid()` ルーチンの詳細は、リファレンス・ページの `pw_mapping(3)` で説明されています。
- ❻ `acl_set_tag_type()` 関数は、指定された ACL エントリのタイプを設定します。現在のタグ・タイプは次のとおりです： `ACL_USER_OBJ` (所有者許可ビット)、`ACL_GROUP_OBJ` (グループ許可ビット)、`ACL_OTHER_OBJ` (その他のユーザの許可ビット)、`ACL_USER` (指定されたユーザに対する許可)、`ACL_GROUP` (指定されたグループに対する許可)。
- ❼ `acl_set_qualifier()` 関数は、タグ・タイプ `ACL_USER` および `ACL_GROUP` の ID を設定します。これは、エントリが参照しているユーザまたはグループを指定するものです。その他のユーザのタグ・タイプには ID は必要ありません。

- ❶ gr\_grouptoid() ルーチンは、グループ名からグループ ID への最適化されたマッピングを提供し、Base セキュリティまたは Enhanced セキュリティのうち使用可能なほうを扱います。詳細は、リファレンス・ページの pw\_mapping(3) に説明されています。
- ❷ acl\_valid() ルーチンは、足りないエントリや重複したエントリがないかどうかチェックします。

## 7.6 ACL の継承例

この節では、プログラムで省略時のアクセス ACL をディレクトリに設定する方法を示し、そのディレクトリ内にファイルおよびディレクトリが作成されるときにどうなるかを説明しています。省略時の ACL には、別のタイプがあり、省略時のディレクトリ ACL と呼ばれます。ディレクトリに省略時のディレクトリ ACL がある (省略時のアクセス ACL もあることもある) 場合、ACL は異なる方法で継承されます。ACL 継承規則の詳細については、『セキュリティ管理ガイド』を参照してください。

/usr/john/acl\_dir ディレクトリに、次のようなアクセス ACL と省略時のアクセス ACL があると仮定します。

```
% getacl /usr/john/acl_dir

# file: /usr/john/acl_dir
# owner: john
# group: prog
#
user::rwx
user:june:r-x
user:fred:r-x
group::rwx
group:mktg:rwx
other::r-x

% getacl -d /usr/john/acl_dir

# file: /usr/john/acl_dir
# owner: john
# group: prog
#
user::rwx
user:june:r-x
user:sally:r-x
group::rwx
group:mktg:rwx
other::rwx
```

次のプログラムを使って、ディレクトリの省略時のアクセス ACL を更新してディレクトリの読み取り/書き込み許可をグループ・エントリから削除し、その後、このディレクトリに通常ファイルとディレクトリを作成できます。下記のコードを `acl_inheritance.c` というファイルに抽出した場合は、次のコマンドを使ってコンパイルできます。

```
% cc -o acl_inheritance -lpacl -lsecurity acl_inheritance.c

#include <unistd.h>
#include <sys/types.h>
#include <prot.h>
#include <errno.h>
#include <sys/acl.h>

#define REGULAR_FILE "regular"
#define DIRECTORY_FILE "dir"

main (argc, argv)
int argc;
char *argv[];
{
    acl_permset_t  acl_permset;
    gid_t          *qualifier = NULL;
    acl_tag_t      tag_type;
    acl_t          acl;
    acl_entry_t     acl_entry;
    gid_t          my_gid;
    int            ret;
    char           pathname[PATH_MAX + 1];
    int            fd;

    /* Did the user enter a directory name and group name? */
    if (argc != 3) {
        printf("Usage: %s directory group\n", argv[0]);
        exit(1);
    }

    /* Map the group name to a gid */
    my_gid = gr_nametoid(argv[2]);
    if (my_gid == (gid_t) -1) {
        printf("No translation for group %s\n", argv[2]);
        exit(1);
    }

    /* Read the default ACL from the directory */
    acl = acl_get_file(argv[1], ACL_TYPE_DEFAULT);
    if (!acl) {
        if (errno) {
            perror("acl_get_file");
        }
    }
}
```

```

        else {
            printf("No default ACL found on %s\n", argv[1]);
        }
    }
    exit(1);
}

ret = acl_first_entry(acl);
if (ret) {
    perror("acl_first_entry");
    exit(1);
}

/* Scan the ACL looking for the entry */
while (acl_entry = acl_get_entry(acl)) {

    /* retrieve the entry type */
    ret = acl_get_tag_type(acl_entry, &tag_type);
    if (ret) {
        perror("acl_get_tag_type");
        exit(1);
    }

    if (tag_type != ACL_GROUP) continue;

    qualifier = (gid_t *)acl_get_qualifier(acl_entry);
    if (!qualifier) {
        perror("acl_get_qualifier");
        exit(1);
    }

    /* Check for appropriate entry */
    if (*qualifier != my_gid) continue;

    ret = acl_get_permset(acl_entry, &acl_permset);
    if (ret) {
        perror("acl_get_permset");
        exit(1);
    }

    *acl_permset = *acl_permset & ~(ACL_READ | ACL_WRITE);

    ret = acl_set_permset(acl_entry, acl_permset);
    if (ret) {
        perror("acl_set_permset");
        exit(1);
    }

    ret = acl_set_file(argv[1], ACL_TYPE_DEFAULT, acl);
    if (ret) {
        perror("acl_set_file");
    }
}

```

```

        exit(1);
    }

    break;

    }

    if (!acl_entry) {
        if (errno) {
            perror("acl_get_entry");
        }
        else {
            printf("ACL entry for %s not found\n", argv[2]);
        }
    }
    exit(1);
}

/* Create the regular file */
sprintf(pathname, "%s/%s", argv[1], REGULAR_FILE);

fd = creat(pathname, 0644);
if (fd == -1) {
    perror("creat");
}
exit(1);
close(fd);

/* Create the directory */
sprintf(pathname, "%s/%s", argv[1], DIRECTORY_FILE);

ret = mkdir(pathname, 0700);
if (ret == -1) {
    perror("mkdir");
}
exit(1);
}
}

```

前述のサンプル・プログラムを実行すると、上に示した省略時の ACL の `mktg` グループから、読み取り/書き込み許可が削除されます。プログラムはその後、このディレクトリに通常ファイルとディレクトリを作成して、ACL が継承されたことを示します。サンプル・プログラムを実行するには、次のコマンドを入力します。

```
% ./acl_inheritance /usr/john/acl_dir mktg
```

上記のコードを実行すると、新たに作成されたファイルのアクセス ACL と、新たに作成されたディレクトリのアクセス ACL および省略時のアクセス ACL は、次のようになります。

```
% getacl /usr/john/acl_dir/regular
# file: /usr/john/acl_dir/regular
# owner: john
# group: prog
#
user::rw-
user:june:r-x
user:sally:r-x
group::r--
group:mktg:--x
other::r--
```

所有しているユーザ，所有しているグループ，およびその他のアクセス許可には，省略時のアクセス ACL と，`creat()` 呼び出しで指定されたモードの論理和が設定されます。ACL が継承されるときには，`umask` は使用されません。その他のエントリは，親ディレクトリの省略時のアクセス ACL に基づいて設定されます。

```
% getacl /usr/john/acl_dir/dir
# file: /usr/john/acl_dir/dir
# owner: john
# group: prog
#
user::rwx
user:june:r-x
user:sally:r-x
group:---
group:mktg:--x
other:---
```

省略時のアクセス ACL を持つディレクトリ内に作成されるサブディレクトリの継承規則は，ファイルの規則と同じです。これは，親ディレクトリに，省略時のアクセス ACL のほかに省略時のディレクトリ ACL がない場合のみ適用されます。

次のコマンド行は，省略時のアクセス ACL を表示します。

```
% getacl -d /usr/john/acl_dir/dir
# file: /usr/john/acl_dir/dir
# owner: john
# group: prog
#
user::rwx
user:june:r-x
user:sally:r-x
group::rwx
group:mktg:--x
other::rwx
```

省略時の ACL は，ディレクトリの親から継承されます。



Generic Security Service Application Program Interface (GSS-API) 関数を使用すると、分散ネットワーク環境にあるアプリケーションは、ネットワーク上で次のセキュリティ・サービスを使用できるようになります。

- 認証 — アプリケーションはユーザまたはサービスの ID を確認できます。
- 完全性 — アプリケーションは、メッセージを受け取るときに、メッセージの不正な変更や破損を検出できます。
- 機密性 — メッセージを暗号化して、送信時に盗聴者に解読できない形に表現できます。

この章では、次の内容について説明しています。

- GSS-API の概要
- Application Security SDK
- Application Security SDK 関数
- ベスト・プラクティス
- 移植性のあるアプリケーションの構築

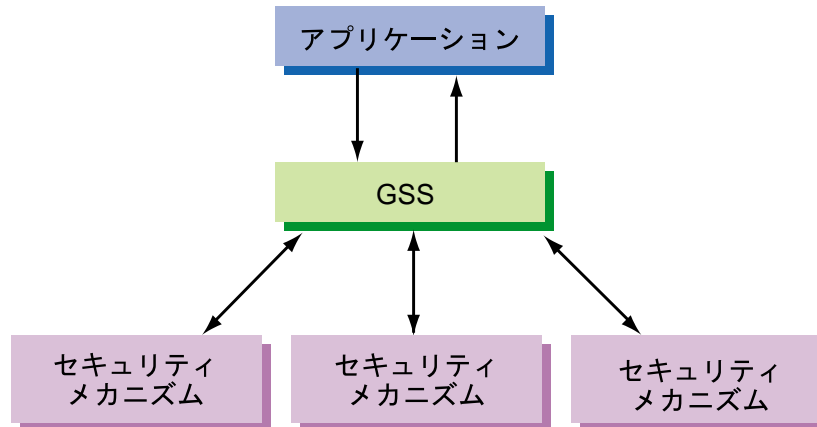
## 8.1 GSS-API の概要

GSS-API は、分散型アプリケーションの保護に使用できる一連の汎用 C 関数を定義する、標準的なプログラミング・インタフェースです。GSS-API には、その処理には欠かせない 2 つの大きな設計上の目標があります。

- セキュリティ・メカニズムからの独立
- トランスポート・プロトコルからの独立

GSS-API はオープン・スタンダードであるため、セキュリティ技術やネットワーク技術が進化するたびに API を変更する必要が生じないように、汎用性を持たせて設計されています。

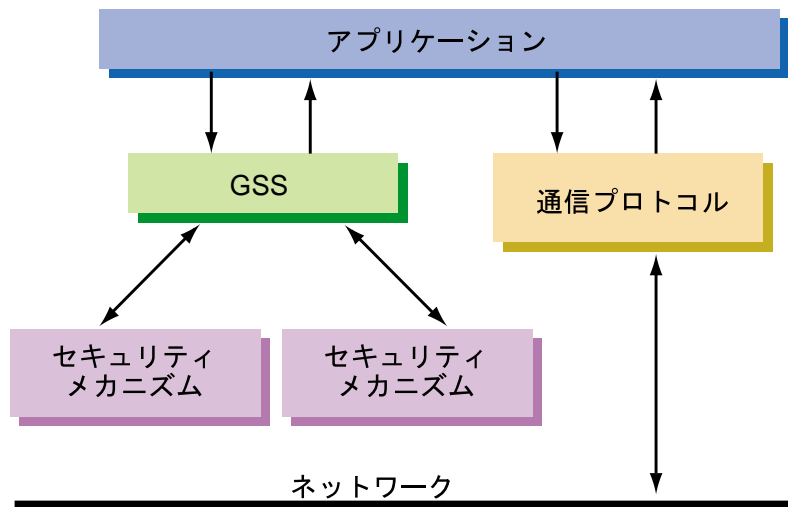
GSS-API は、次のようなアーキテクチャを使って、基盤のセキュリティ・メカニズムと技術を幅広くサポートしています。



ZK-1825U-AI

セキュリティ・メカニズムとは、セキュリティを提供する手段のことです (たとえば Kerberos や公開鍵暗号化など)。使用されている暗号化技術だけでなく、その技術が使用しているデータの構文と意味も含みます。GSS-API 標準を使って保護されたアプリケーションは、1 つまたは複数のセキュリティ・メカニズムを使用できます。

GSS-API は幅広いネットワーク環境で使用できます (たとえば、TCP/IP、SNA、DECnet)。GSS-API は、トランスポート・メカニズムを提供するように設計されたものではありません。その代わり、任意のネットワーク・トランスポート上でセキュリティを提供する設計になっています。トランスポートはアプリケーションが提供する必要があります。通信プロトコルは、プロセス間通信の経路または一連のネットワークが考えられます。



ZK-1826U-AI

GSS-API 関数はアプリケーションに情報を戻し、アプリケーションは使用中の通信プロトコルを通じてその情報を送信します。分散型アプリケーションのもう一方の側は、情報を GSS-API ライブラリに渡します。

GSS-API 標準を使ってアプリケーションを保護する開発者のために、メカニズムとトランスポートからの独立という設計目標が、基盤となるハードウェアおよびソフトウェア・プラットフォームから独立した一貫性のあるインタフェースを提供しています。つまりプログラミングの投資は 1 回だけで済むということです。アプリケーションを保護するための変更に対する投資は、技術が進化したとしても一定のままです。

### 8.1.1 GSS-API の前提条件

GSS-API 標準は次のような前提条件に基づいています。

- アプリケーションは分散されている。

GSS-API 標準は、アプリケーションは分散ネットワーク・アプリケーションであるか、ピア・ツー・ピアまたは開始側/受け入れ側の関係を使って 2 つの部分に分かれているものと想定しています。

- ソース・コードは変更可能である。

GSS-API 標準は、GSS-API 関数をアプリケーションのソース・コードに組み込めるものと想定しています。

- アプリケーションはトークンの配信を保証する。

トークンとは、GSS-API によって戻され、アプリケーションとそのピアとの通信に必要な不透過なデータ・オブジェクトです。GSS-API 標準は、アプリケーションはコンテキストが確立してから終了するまでに生成されたトークンを、生成された順番で配信できるものと想定しています。

- アプリケーションはそのデータ・オブジェクトの割り当てを解除する。

GSS-API 標準は、アプリケーションがデータ・オブジェクトを割り当てた場合は、割り当ての解除もそのアプリケーションが行うものと想定します。GSS-API 関数によってデータ・オブジェクトが戻されると、アプリケーションは必ず、対応する GSS-API 関数を使用してそのオブジェクトを解放しなければなりません。その結果オブジェクトの割り当ては解除されます。そうしないと、アプリケーションでメモリ・リークやメモリ障害が発生する可能性があります。割り当ての解除に正しい関数を使用しないと、セキュリティ・ネットワークへの侵入を許すような状況を作ることになります。

### 8.1.2 詳細情報

GSS-API は、現在進行中の Internet RFC (Request For Comments) プロセスの一部として策定された業界標準です。

Application Security SDK は、次の RFC に基づいています。

- RFC 2078 “ Generic Security Service Application Program Interface, Version 2, Update 1 ” 1998 年 9 月 3 日
- “ Generic Security Service API Version 2: C-bindings, ” 1998 年 8 月 7 日
- RFC 1964 “ The Kerberos Version 5 GSS-API Mechanism, ” 1996 年 6 月
- RFC 1510 “ The Kerberos Network Authentication Service (V5), ” 1993 年 9 月
- Internet Draft “ Public Key Cryptography for Initial Authentication in Kerberos ” (updates RFC 1510)

ファイル名: draft-ietf-cat-kerberos-pk-init-07.txt

GSS-API 標準は、作業グループ Common Authentication Technology Internet Engineering Task Force (CAT-IETF) の監督下にあります。GSS-API についての詳細は、IETF の Web サイト、[www.ietf.org](http://www.ietf.org) を参照してください。

## 8.2 Application Security SDK

HP は、Application Security Service Developers Kit (SDK) を通じて GSS-API 関数を実装しています。Application Security SDK は、GSS-API 標準にユニークな機能をいくつか追加した、GSS-API バージョン 2.0 の実装です。

- Application Security SDK は、Kerberos 5 のメカニズムをサポートしています。サンプル・コードとドキュメントに、Kerberos 5 と GSS-API を使って分散型アプリケーションを保護するための明確な手順と、このセキュリティ・メカニズムに固有の考慮事項を示してあります。
- Application Security SDK には、GSS-API の機能を強化する、追加の関数も多数含まれています。エクステンションと呼ばれるこれらの関数では、次のような、Kerberos 5 の追加のセキュリティ機能の使用が可能です。
  - 認証トークンとユーザ・データの暗号化を含む、Triple DES (DES3) の完全サポート。HP では、すべてのデータを DES3 を使って暗号化することをお勧めします。DES3 は、DES に比べてセキュリティを大幅に向上させます。
  - 開始側のアプリケーションがセキュリティ・コンテキストを確立する前に必要な、初回信任状を取得するためのプログラミング・サポート。このエクステンション関数を使うと、初回信任状を取得するときに、*kinit*、*ActiveTRUST SignOn* (CyberSafe Corporation から入手可能)、HP の *Single SignOn* などを使う必要がなくなります。
  - トークン・カードなど、ハードウェア認証デバイスの認識。これは認証要求の際に、クライアント・レベルでセキュリティのレベルをさらに 1 つ追加します。
  - Kerberos 5 信任状を延長する能力。これは、既存の有効な信任状存続期間を延ばし、ユーザがユーザ名とパスワードを提示しなければならない回数を減らします。

## 8.3 Application Security SDK 関数

Application Security SDK 関数は複数のカテゴリに分類できます。分散型アプリケーションの保護に必要なのは 1 つのサブセットだけです。これらの関数は、以下のカテゴリに従って分けることができます。

- 名前管理関数 — GSS-API で使われる内部と外部の名前を扱う関数。
- 信任状管理関数 — 信任状の取得、問い合わせ、解放に使われる関数。このカテゴリには、Application Security SDK の中にあり、GSS-API 標準ではない HP エクステンション関数も含まれています。このカテゴリの関数は初回認証の実装に使われており、DES3 暗号化、ハードウェア認証、信任状の延長のためのサポートが含まれています。
- セキュリティ・コンテキスト管理関数 — セキュリティ・コンテキストの開始、受け入れ、エクスポート、インポート、問い合わせ、削除に使われる関数。
- メッセージ関数 — データの完全性、データの発信元、必要に応じて機密性の保護に使われる関数。
- その他の関数 — 状態の表示、バッファの解放、オブジェクト識別子 (OID) セットの操作に使われる、その他のサポート関数。
- V1 準拠関数 — GSS-API Version 1 との相互運用性のためにサポートされている関数。これらの関数は GSS-API Version 2 に置き換えられています。

### 8.3.1 名前管理関数

ユーザがネットワーク上のシステムにログインするときにユーザ名を使うのと同じように、GSS-API もまたネットワーク内のエントリの識別に名前を使います。ユーザ名は、システムの命名構造に従ってユーザを識別するものです。GSS-API 標準では、名前はアプリケーションまたはそれを使うユーザを識別します。

Kerberos 5 メカニズムでは、名前はプリンシパルに変換されます。プリンシパルとは、Kerberos と何らかの秘密 (通常はパスワード) を共有する任意のユーザ、ネットワーク・サービス、アプリケーション、システムを指します。プリンシパルは、レルム内で一意の名前と、対応する鍵を持っている必要があります。

GSS-API では、外部名、エクスポート名、内部名、メカニズム名の 4 つの形式の名前を使います。形式が異なる名前の中で、変換を行うための関数も用意されています。

- 外部名は、GSS-API 標準への入力として使われるテキスト文字列です。
- エクスポート名は、標準フォーマットで表したオクテット文字列であり、GSS-API 関数によって生成され、GSS-API 標準の外部での名前比較に使われます。
- 内部名は不透過です。つまり、エラー・メッセージなどに名前を表示できないということです。この種類の名前は、GSS-API 内部のあらゆる目的のために使われます。Kerberos 5 メカニズムとの使い方の例としては、特定のユーザ名に属する信任状を見つけるなどがあります。
- メカニズム名は、内部名のうち、メカニズムに固有の特殊なケースです。つまり、1 つのメカニズムに特有であることを意味します。Application Security SDK のように、1 つのメカニズムだけをサポートしている場合は、内部名とメカニズム名は同じです。

GSS-API では、名前は次の目的で使用されます。

- 信任状の取得 — 一方のアプリケーションが信任状を取得するために、その名前を GSS-API の内部フォーマットにインポートします。
- セキュリティ・コンテキストの確立 — 開始側は、セキュリティ・コンテキストを確立する相手を名前で識別します。
- 名前の比較 — セキュリティ・コンテキストが確立された後、名前の比較が必要になることがあります。名前の比較は、正しい名前フォーマットが使われている限り、GSS-API 標準の内部と外部のどちらでも可能です。
  - GSS-API 標準の内部では、名前の比較に内部名を使います (`gss_compare_name()` を使う)。
  - GSS-API 標準の外部では、名前の比較にはエクスポート名を使う必要があります。たとえば受け入れ側は、開始した相手の名前を、アクセス制御リスト (ACL) と突き合わせて比較するなどが考えられます。この比較を実行するための関数呼び出しは、この API には提供されていません。

アプリケーションまたはそれを使うユーザの名前は、GSS-API によって定義されている構文、構造、命名規則に従ったものではありません。アプリ

ケーションの名前は、アプリケーション自身に依存しています。GSS-API は汎用であるため、異なるフォーマットで名前を提供できます。名前を表現するためのこれらの手法は、名前タイプと呼ばれ、オブジェクト識別子 (OID) として渡されます。

アプリケーションは、その名前として GSS-API の省略時値を使うこともできます。省略時の名前を使う場合は、特定の名前を指定する必要がなく、OID を渡す必要もありません。この場合、名前は GSS-API の実装に依存します。

GSS-API 関数	説明
<code>gss_canonicalize_name()</code>	内部名をメカニズム固有の名前に変換します。
<code>gss_compare_name()</code>	内部フォーマットによる 2 つの名前が同じかどうかを比較します。
<code>gss_display_name()</code>	人間が解読できる形式、あるいは表示可能な形式に名前を変換します。名前のフォーマットは、GSS-API の実装に固有です。このため、 <code>gss_display_name()</code> 関数から戻される表示可能な形式と、そうでない名前は比較しないでください。比較に必要な名前を生成するには、関数 <code>gss_export_name()</code> を使います。この名前は、 <code>gss_import_name()</code> を使ってインポートすることはできません。  この関数によって生成される表示可能な形式は、バッファに置かれます。バッファは、 <code>gss_release_buffer()</code> を呼び出すことによって解放しなければなりません。
<code>gss_duplicate_name()</code>	既存の内部形式の名前のコピーを作成します。
<code>gss_export_name()</code>	内部名を、標準のフォーマットのエクスポート形式 (文字列) に変換します。



GSS-API 関数	説明
<code>gss_import_name()</code>	<p>アプリケーションの名前 (外部形式) を内部形式に変換します。アプリケーションは、その名の解析方法を指定するオブジェクト識別子 (OID) を渡します。たとえば、UNIX の <code>uid</code> またはログイン名などが考えられます。</p> <p>名前は構造体に入れられて戻され、<code>gss_release_name()</code> を呼び出すことによって解放しなければなりません。</p> <p><code>gss_import_name()</code> には、省略時の名前は指定できません。省略時の名前を指定できるのは、<code>gss_acquire_cred()</code> で信任状を取得するときのみです。</p>
<code>gss_inquire_mechs_for_name()</code>	<p>指定された名前タイプをサポートするメカニズムを識別します。Application Security SDK では、これは常に Kerberos 5 になります。</p>
<code>gss_inquire_names_for_mech()</code>	<p>指定されたメカニズムによってサポートされている名前タイプの一覧を表示します。Application Security SDK では、これは常に Kerberos 5 がサポートしている名前タイプになります。</p>
<code>gss_release_name()</code>	<p><code>gss_import_name()</code> が呼び出されたときに割り当てられた、GSS-API 名の記憶領域を解放します。</p>

### 8.3.1.1 省略時の名前と構文

Kerberos のプリンシパル名は、次の形式をとります。 `name/instance@REALM` (`/instance` および `@REALM` は省略可能)。`/instance` を省略した場合は、空のインスタンスであるとみなされます。`@REALM` を省略した場合は、省略時のレルムであるとみなされます。`/` で区切ることによって、複数のインスタンスを指定できます。

Kerberos 5 メカニズムを使った信任状を取得するときに、アプリケーション名に GSS-API の省略時値が使われている場合、使用される名前は、キャッシュがあれば、省略時の信任状キャッシュにある省略時の信任状の省略時のプリンシパルになります。キャッシュがなければ、省略時の名前は次のルールに従って決定されます。

- ユーザ (つまり人間) の省略時のプリンシパルは、ユーザのログイン名です。
- サービスの省略時のプリンシパルは、`host/fqdn@REALM` です (`fqdn` は、サービスが実行しているシステムの完全修飾ドメイン名、`REALM` はシステムの省略時のレルム)。

HP では、セキュリティを高めるために、サービスには省略時のプリンシパル名 `host/` を共有するのではなく、独自のプリンシパル名を付けることをお勧めします。たとえば、セキュアな FTP サービスは、プリンシパル名 `ftp/fqdn@REALM` または `ftp@REALM` (`fqdn` インスタンスを省略) を使うことが考えられます。セキュアな telnet サービスはプリンシパル名 `telnet/fqdn@REALM` または `telnet@REALM` を使うことが考えられます。

アプリケーションが `gss_import_name()` を呼び出すときに、省略時の入力フォーマットを使用しなかった場合、アプリケーションは、その名前の解析方法を指定するオブジェクト識別子 (OID) を渡します。たとえば、`GSS_KRB5_NT_HOSTBASED_SERVICE_NAME` OID は、`service@host` という形式の名前の解析構文を指定します。このパラメータの `NT` は、名前タイプを表します。

### 8.3.2 信任状管理関数

信任状は、アプリケーションがその ID を証明するために使います。分散型アプリケーションの一方は、セキュリティ・コンテキストの開始と受け入れのために使われる信任状を取得する必要があります。

Kerberos 5 では、3 つのタイプの信任状があります。

- 初回チケット — このタイプの信任状は、ユーザ・プリンシパルの初回認証の後、Kerberos セキュリティ・サーバから受け取ります。初回チケットは正しくはチケット・グランティング・チケット (または TGT) と呼ばれ、開始側のアプリケーションに必要なになります。
- サービス・チケット — このチケットは、保護されているアプリケーションに、ユーザ・プリンシパルを使ってアクセスすることを許可するものです。サービス・チケットを受け取るには、TGT を取得しておく必要があります。
- サービス鍵テーブル・エントリ — プリンシパル・データベース中のプリンシパルと関連付けられた、秘密鍵のコピーです。この鍵は、データ

ベース管理プログラムを使って、プリンシパル・データベースからサービス鍵テーブル・ファイルへ抽出しなければなりません。サービス鍵テーブル・ファイルの鍵は、初回チケットの取得の代わりに、受け入れ側のアプリケーションによって使われます。

Kerberos 5 のセキュリティ用語とインフラストラクチャの詳細は、「セキュリティの基本」を参照してください。

Kerberos 5 メカニズムと GSS-API 標準を使う場合、初回チケットは、身元を証明するものとして、Kerberos プリンシパルによって保持され、提示されます。これらの信任状は、初回認証のためにトラステッド KDC に接続されたことを証明するために、プリンシパルによって使用されます。通常、セキュリティ・サーバによって提供される初回チケットは、信任状キャッシュと呼ばれる中央の格納場所に格納されます。

GSS-API では、初回チケットは記憶領域から取得され、セキュリティ・コンテキストを確立するために使われます。標準 GSS-API と Kerberos 5 メカニズムでは、開始側のアプリケーションは、`gss_acquire_cred()` の呼び出しの前に、初回チケット (TGT) を取得していなければなりません。ユーザは、`kinit` またはこれに相当するアプリケーション (たとえば、`Single SignOn`) を実行することによって初回チケットを取得できます。受け入れ側のアプリケーションは、サービス鍵テーブル・ファイルに格納されている鍵を使って、その ID を確認します。

Application Security SDK は、HP エクステンション関数と呼ばれる追加の関数を提供しています。この関数を使って、次のような各種のタスクを実行できます。

- コンテキストの開始前に、初回信任状を取得する。これは、ユーザから秘密情報を取得するプログラム・プロンプトを通じて、または、開始側のアプリケーションにも開始側のホスト上のサービス鍵テーブル・ファイルに格納されている秘密鍵の使用を許可することによって行われます。
- スマート・カード・デバイスから公開鍵信任状を取り出して、TGT を取得する。
- 転送可能、または延長可能など、特定のオプションを付けて初回信任状を要求する。
- 信任状の存続期間を延ばすために、有効期限前に信任状を延長する。
- 不要になったら信任状に関連付けられているバッファを解放する。

これらの特殊な HP エクステンション関数を使うと、初回信任状が呼び出し側のアプリケーションによって受け取られ、信任状キャッシュに置かれます。その後、この初回信任状を GSS-API 関数を使って取得し、セキュリティ・コンテキストを確立するために必要なサービス・チケットを取得する目的で使用できます。

GSS-API 関数	説明
<code>gss_acquire_cred()</code>	初回信任状を、アプリケーションが使うために格納場所から取り出します。コンテキストを確立するためにはこの手順が必要となります。信任状は、この関数を呼び出す前に存在している必要があります。
<code>gss_add_cred()</code>	信任状を追加的に作成します。この関数は、複数のメカニズムがサポートされている場合に使われます。1つのメカニズムだけを実装している場合は、HP は、関数 <code>gss_acquire_cred()</code> を使うことをお勧めします。
<code>gss_inquire_cred()</code>	信任状についての情報を戻します (たとえば、この信任状がサポートしているメカニズム、信任状タイプ、信任状の存続期間の一覧など)。Kerberos 5 の場合、これは初回信任状に関する情報になります。
<code>gss_inquire_cred_by_mech()</code>	特定のメカニズムに固有の既存の信任状に関する情報を取得します。Application Security SDK の場合、この情報は、Kerberos セキュリティ・サーバから取得した TGT から取得されます。
<code>gss_release_cred()</code>	使い終わった信任状をアプリケーションから解放します。 <code>gss_acquire_cred()</code> または <code>gss_add_cred()</code> で取得した信任状は、すべてこの関数を使って解放する必要があります。Kerberos 5 の場合、これは信任状キャッシュにある初回信任状エントリには影響を及ぼしません。

これらの関数は、認証サーバやネットワーク・ファイル・システムなどのネットワーク・エンティティ間で、保留中のやり取りをブロックできます。ブロックは、オペレーティング・システムの機能と、使用されているセキュリティ・メカニズムに依存します。Kerberos 5 メカニズムでは、信任状管理関数は、ディレクトリや認証サーバなど、他のネットワーク・エンティティの間の保留中のやり取りをブロックできます。

### 8.3.2.1 初回信任状の取得

GSS-API 標準では、コンテキストの確立前に開始側のアプリケーションは、初回信任状を取得し、これを信任状キャッシュと呼ばれる中央の格納場所におく必要があります。格納された信任状は、関数 `gss_acquire_cred()` を使ってキャッシュから取得されます。

#### 8.3.2.1.1 開始側のアプリケーション

Application Security SDK は、開始側のアプリケーションの初回認証を実行する特別な関数を提供しており、GSS-API 標準を使って取り出せるように信任状をキャッシュに格納します。関数 `csf_gss_acq_user()` を、Single SignOn, Credentials Manager, kinit などによって制御される認証処理の代わりに使用できます。

この関数に指定するパラメータは秘密タイプであり、本人かどうかを証明するためにプリンシパルから取得されます。ユーザ・プリンシパルの場合、パラメータとして渡す秘密は通常はプリンシパル名とパスワードであり、プロンプトから指定しますが、サービス鍵テーブル・エントリでも構いません。

初回チケットまたは TGT は、転送可能チケットや代理可能チケットなどの属性を指定し、チケットの存続期間と延長期限を指定することで、選択した Kerberos チケット・フラグを付けて取得できます。

#### 8.3.2.1.2 受け入れ側のアプリケーション

受け入れ側のサービス・プリンシパルは、サービス鍵テーブル・ファイルを使う必要があります。このファイルは、Kerberos データベース管理プログラムによって最初に生成され、提供されたランダム鍵の形式の、サービス・プリンシパルの秘密を格納するために使われます。このタイプの初回認証は、GSS-API 関数 `gss_acquire_cred()` を使って実行されます。

#### 8.3.2.1.3 DES3

Application Security SDK は、DES3 暗号化のためのサポートを提供しています。DES3 を使うことで、DES 暗号化よりも大幅に改善されたセキュリティ保護を実現しています。

DES3 暗号化を使ってメッセージを暗号化できるようにするには、次の条件が満たされている必要があります。

- *Active*TRUST Security Server が DES3 向けに設定されていること。

- 開始側と受け入れ側のアプリケーション用のプリンシパルが、プリンシパル・データベースの中に同じ DES3 鍵を持っていること。
- 開始側のプリンシパルが DES3 を使って TGT を取得していること。
- 開始側のアプリケーションが、セキュリティ・コンテキストの開始時に DES3 の使用を指定していること。

---

注意

---

単一のセキュリティ・コンテキストに対して複数の暗号化システムは許可されていません。どれか 1 つだけを使う必要があります。

---

### 8.3.2.2 信任状の属性

関数 `gss_inquire_cred()` を使って、特定の信任状に関連付けられているプロパティ、または属性を確認できます。この関数によって戻される Kerberos 情報には、信任状の存続期間、サポートされているメカニズム、信任状の使用方法などが含まれています。初回信任状に割り当てられる属性は、TGT を取得するときに要求された属性によって異なります。特定の TGT を使って取得したサービスは、その TGT に割り当てられている属性を継承します。

信任状には有効期限が定義されていることがあり、これを過ぎるとその信任状は無効となります。`gss_inquire_cred()` 関数を使って、信任状が期限切れになったかどうかを判別できます。

メカニズムの実装によっては、信任状が期限切れになった後でアプリケーションがそれを使うことはできない場合があります。その場合、アプリケーションは信任状を解放し、もう一度信任状を取得してから継続することになります。Kerberos 5 メカニズムでは、信任状の存続期間は、Kerberos チケットの存続期間として定義されています。信任状の有効期限が切れた後は、アプリケーションはそれを使って新しいセキュリティ・コンテキストを確立することはできません。アプリケーションは、`csf_gss_acq_user()`、`gss_acquire_cred()` を使って新しい信任状を解放・取得する必要があります。

---

## 注意

---

GSS-API 標準は、最初に確立された存続期間を過ぎた信任状の期限を延長するための関数は提供していません。しかし、有効期限が切れていない場合は、HP エクステンション関数 `csf_gss_renew_cred()` を使って信任状の期限を延長できます。信任状の期限を延長するためには、初回チケットはセキュリティ・サーバから延長可能属性フラグを付けて要求されたものでなければなりません。

---

### 8.3.2.3 信任状の格納場所

Kerberos 5 では、信任状は、ローカル・ホストのハードディスクまたは他の永続的な記憶媒体に格納されていると想定しています。

ユーザ・プリンシパルは、環境変数によって指定されていない限り、信任状キャッシュの省略時の格納場所に格納されているものと想定されます。場所はプラットフォームによって異なります。

UNIX ユーザの省略時の信任状キャッシュ・ファイルは、環境変数 `CSFC5CCNAME` が他のパス名に設定されていなければ、`krb5/tmp/cc/krb5cc_uid` (`uid` はパスワード・ファイルから取り出したユーザ ID 番号) になります。

サービス・プリンシパルの場合、省略時のサービス鍵テーブル・ファイルは、プラットフォームごとに異なる場所にあります。たとえば UNIX システムの場合、省略時のファイルは、`CSFC5KTNAME` 環境変数が別のパス名に設定されていなければ、`/krb5/v5srvtab` になります。

これらの信任状キャッシュとサービス鍵テーブル・ファイルの設定についての詳細は、SSO の『*Installation and Administration Guide*』を参照してください。

### 8.3.2.4 信任状関連リソースの管理

`gss_acquire_cred()` で取得した信任状はすべて、`gss_release_cred()` を呼び出すことによって解放する必要があります。信任状が解放されるときの実際の動作は、使用するセキュリティ・メカニズムによって異なります。いくつかのケースでは、信任状の内部バッファだけが解放されます。その他

のケースでは、ディスクに格納されている信任状も解放されます。Kerberos 5 メカニズムでは、この関数では内部バッファだけが解放されます。

信任状をいつ解放するかを決める前に、アプリケーションは、実行中にセキュリティ・コンテキストを何回か開始したり、受け取ったりする予定があるかどうかを判断しなければなりません。予定がある場合、アプリケーションは、信任状を取得してこれを一度使った後、すぐに解放するのではなく、後でセキュリティ・コンテキストの開始関数または受け入れ関数を使って再使用する必要があります。Kerberos 5 の実装では、こうすることで性能を向上させています。

8.3.3 セキュリティ・コンテキスト管理関数

セキュリティ・コンテキストは、一度確立されると、分散環境にあるアプリケーション間で一意の対話を定義します。セキュリティ・コンテキストは、一方のアプリケーションによって開始され、もう一方のアプリケーションによって受け取られます。セキュリティ・コンテキストは、保護対象のメッセージの交換を行う前に確立する必要があります。

GSS-API 標準の Kerberos 5 の実装では、コンテキストが確立するときに、サービス・チケットの管理に必要な Kerberos のプロセス実行するようになっています。そのプロセスで、サービス・チケットの取得、転送、確認が行われます。コンテキストが確立されると、そのコンテキストには鍵とその他のセキュリティ・パラメータが付加します。セキュリティ・コンテキストの内容は、メッセージを保護するために直接的に使われます。

GSS-API 関数	説明
<code>csf_gss_get_context_options ( )</code>	使用されている暗号化の種類の確認も含め、既存のセキュリティ・コンテキストに関する情報を取得します。これは Application Security SDK に含まれている HP のエクステンション関数です。
<code>gss_accept_sec_context ( )</code>	開始側のアプリケーションからのセキュリティ・コンテキストの確立要求を受け取ります。
<code>gss_context_time ( )</code>	セキュリティ・コンテキストの存続期間がどれだけ残っているかを示します。存続期間の決め方とその意味は、メカニズムに依存します。



GSS-API 関数	説明
<code>gss_delete_sec_context()</code>	不要になったコンテキストを破壊します。開始、または受け入れが終わったセキュリティ・コンテキストは、それぞれこの関数を使って破壊する必要があります。
<code>gss_export_sec_context()</code>	セキュリティ・コンテキストを別のプロセスに転送します。
<code>gss_import_sec_context()</code>	エクスポートされたセキュリティ・コンテキストをインポートします。
<code>gss_init_sec_context()</code>	受け入れ側アプリケーションとのセキュリティ・コンテキストを開始します。
<code>gss_inquire_context()</code>	セキュリティ・コンテキストに関する情報を取得します。

これらの関数は、アプリケーションとセキュリティ・サーバの間で保留中になっているやり取りをブロックすることがあります。

#### 8.3.3.1 メカニズムの識別

GSS-API 標準は、使用可能なセキュリティ・メカニズムが複数あるような環境で使われるように設計されています。つまり、対象となるセキュリティ・メカニズムをプログラムの識別しなければならないということです。

`gss_init_sec_context()` には、パラメータの 1 つとして、開始側が使おうとしているセキュリティ・メカニズム識別子 (または `mech_type`) を渡します。メカニズム識別子は、オブジェクト識別子 (OID) として渡されます。または、開始側は省略時のメカニズム識別子を使うこともできます。省略時のメカニズムとして何が選択されるかは、GSS-API の実装に依存します。Application Security SDK の場合、省略時のメカニズムは Kerberos 5 です。

#### 8.3.3.2 トークンの交換

`gss_init_sec_context()` の呼び出し時に、Application Security SDK はユーザの信任状キャッシュにサービス・チケットがあるかどうか検索します。キャッシュにサービス・チケットがなければ、Application Security SDK は KDC からサービス・チケットを取り出します。戻されたサービス・チケットは、ユーザの信任状キャッシュに格納されます。GSS-API は、このサービス・チケットとそれに付随する情報をコード化し、これをトークンに戻して相手側のアプリケーションに送信されるようにします。

`gss_accept_sec_context()` の呼び出し時に、GSS-API は、サービス鍵テーブル・ファイル中のその秘密鍵を使ってこのトークンを復号し、サービス・チケットを確認します。確認が成功して相互認証が要求されると、GSS-API は応答の中にトークンを生成します。相互認証が要求されなかった場合、リターン・トークンは生成されません。エラーが発生すると、GSS-API はエラー・トークンを生成します。

### 8.3.3.3 オプションのセキュリティ手段

コンテキストを開始するときに、アプリケーションはいくつかのオプションのセキュリティ手段を指定できます。これらオプションのセキュリティ手段についての GSS-API の実装は、使用されているセキュリティ・メカニズムに依存します。

以下の節では、オプションのセキュリティ手段について説明します。

- チャンネル・バインディング
- 機密性と完全性
- リプレイ検出
- 順序外メッセージ検出
- 相互認証
- 暗号化タイプ：DES と DES3
- 信任状の委任

#### 8.3.3.3.1 チャンネル・バインディング

チャンネル・バインディングは、偽装者によって盗聴されたコンテキスト確立トークンが再使用される可能性があるときに、その有効範囲を限定することによって、コンテキストの確立時に、分散アプリケーション同士の認証の質を高めるために使われます。チャンネル・バインディングは、分散アプリケーションによって指定される追加のデータ項目をセキュリティ・コンテキストに付加することによってこれを実現します。

チャンネル・バインディングは、次の 2 つの部分で構成されます。

- アドレス情報
- オプションのアプリケーション・データ

アドレス情報は、開始側と受け入れ側のどちらの場合もアドレス・タイプとアドレス値からなります。アドレス・タイプは、アドレス値のバッファに格納されているプロトコルのタイプを示します。アドレス値は、開始側または受け入れ側の実際のアドレスです (アドレス・タイプに対応したフォーマットで表現されます)。

Kerberos 5 では、KDC は、サービス・チケットを作成するときにクライアントのアドレスをエンコードしてチケットに埋め込みます。開始側がそのチケットを受け入れ相手に提示し、チャンネル・バインディングが `gss_accept_sec_context()` に渡されると、GSS-API はこのチケットにあるクライアント・アドレスが、チャンネル・バインディングにある開始側のアドレスと一致しているかどうかをチェックします。ただしこのチェックは、TCP 接続を使っている場合は行われません。

---

#### 注意

---

サポートされているアドレス・フォーマットは、`GSS_C_AF_INET` のみです。

---

チャンネル・バインディングのデータ構造体のもう 1 つの構成要素は、アプリケーション・データであり、開始側が受け入れ側に比較してもらうために相手に送信できます。アプリケーション・データは、使用されていれば、アプリケーションによって判別されます。コンテキストを開始するときに指定されるデータは、受け入れ側で完全に一致しなければなりません。たとえば、アプリケーションはその「バージョン番号」をアプリケーション・データに挿入して、バージョン番号が一致しないアプリケーションによるセキュリティ・コンテキストの確立を禁止することができます。

チャンネル・バインディングは、`gss_init_sec_context()` によってトークンとしてエンコードされ、その後で受け入れ側に送信されます。チャンネル・バインディングは、`gss_accept_sec_context()` によってチェックされます。セキュリティ・コンテキストを開始するアプリケーションは、`gss_init_sec_context()` を呼び出す前にチャンネル・バインディングの値を決定する必要があるため、両方のアプリケーションが同一の値をセキュリティ・コンテキスト関数に提示しなければなりません。GSS-API は、受け入れ側のアプリケーションでチャンネル・バインディングをチェックします。

チャンネル・バインディングが `gss_init_sec_context()` に渡されると、このチャンネル・バインディングを使ってハッシュ値が計算されます。ハッシュ値は、トークンにコード化されます。`gss_accept_sec_context()` の呼び出し時に、GSS-API は渡されたチャンネル・バインディングを使ってハッシュ値を計算し、計算したハッシュ値と、渡されたトークンのハッシュ値を比較します。

#### 8.3.3.3.2 機密性と完全性

開始側は、メッセージに機密性か完全性、また両方を持たせ、これをメッセージの送受信時に使うように指定できます。受け入れ側は、メッセージの機密性か完全性、または両方がこのコンテキストに必要なかどうかを問い合わせることができます。この情報は、開始側に戻されます。

---

#### 注意

---

開始側は機密性を要求できますが、受け入れ側がこれをサポートできない場合、機密性は使われません。

---

`gss_init_sec_context()` 関数の `GSS_C_CONF_FLAG` と `GSS_C_INTEG_FLAG` を使って、機密性と完全性の要件を指定します。

#### 8.3.3.3.3 リプレイ検出

リプレイ検出とは、注目しているトークンまたはメッセージが以前に送信されたものかどうかを、一方のアプリケーションがチェックするというものです。

開始側はそのメッセージを、メッセージ・リプレイ・キャッシュと突き合わせてチェックするよう指定できます。これは、メッセージが以前に送信されたメッセージの繰り返しかどうかを判別するために使います。

メッセージがキャッシュされるにつれてより多くのリソースをキャッシュに割り当てるため、リプレイ・キャッシュは必要に応じて拡張します。ただしキャッシュ・サイズには上限があるので注意してください。リプレイ・キャッシュが一定のサイズに達した場合、新しいエントリを 1 つ追加するたびに、古いエントリが 1 つずつ解放されます。キャッシュ全体に関連付けられているリソースは、セキュリティ・コンテキストが削除されるときに解放されます。

---

## 注意

---

Kerberos 5 でサポートされているリプレイ・キャッシュは、2 種類あります。1 つはメッセージ・リプレイ・キャッシュで、コテキスト確立時のパラメータの設定によって制御される、メモリ・ベースのキャッシュです。もう 1 つは認証リプレイ・キャッシュで、UNIX 環境変数で設定が制御されるファイル・ベースのキャッシュです。

---

### 8.3.3.3.4 順序外メッセージ検出

開始側は、メッセージの順番を決めて(または番号付けして)、すべてのメッセージが受信されたかどうかと、それが送信された順番で受信されたかどうかをチェックするように指定できます。メッセージの順序付けは、リプレイ検出機能も提供します。

### 8.3.3.3.5 相互認証

開始側は、受け入れ側にも開始側に対する認証を実行するように指定できます。こうすることで、アプリケーションの双方が、相互に正真正銘の相手とやり取りしているということを確認できます。

### 8.3.3.3.6 暗号化タイプ: DES と DES3

開始側は、初回認証時に DES3 暗号化標準を指定できます。さらに開始側は、セキュリティ・コンテキストを開始するときに、DES3 暗号化も要求できます。DES3 暗号化は、DES 暗号化よりも大幅に改善されたセキュリティを提供しています。

### 8.3.3.3.7 信任状の委任

Kerberos 5 の実装では、開始側は GSS\_C\_DELEG\_FLAG 委任フラグを設定して、初回チケット (TGT) を別のネットワーク・ホストに転送するように指定できます。ただし、TGT を転送できるのは、その TGT に Kerberos の転送可能属性がある場合に限られます。

### 8.3.3.4 対象とするセキュリティ手段の指定

開始側は、`gss_init_sec_context()` を呼び出して、セキュリティ・メカニズムに使わせるセキュリティ手段を指定します。アプリケーションには、

セキュリティ・メカニズムによって提供されるセキュリティ手段を示すフラグが戻されます。GSS-API 標準で定義されているすべてのセキュリティ手段を、セキュリティ・メカニズムが提供できるとは限りません。

セキュリティ・コンテキストの確立時に使うセキュリティ手段を開始側が指定すると、受け入れ側はこれを受け入れなければなりません。この時点では、ネゴシエーションの余地はありません。たとえば、開始側が `gss_init_sec_context()` を呼び出したときに順序付けまたはリプレイ検出を指定した場合、受け入れ側はそれを提供しなければなりません。

開始側が `gss_init_sec_context()` を呼び出したときにチャンネル・バインディングが指定された場合、受け入れ側は、`gss_accept_sec_context()` の呼び出しの中で完全に一致したチャンネル・バインディングを指定しなければなりません。

#### 8.3.4 メッセージ関数

GSS-API メッセージ関数は、メッセージ単位で保護を実行します。メッセージ関数は、どちら側のアプリケーションからも呼び出すことができます。アプリケーションがこれらの関数を呼び出すときには、*QOP* (quality of protection) を指定します。これは、メッセージの処理に使われる暗号化アルゴリズムを識別するものです。QOP の値はメカニズムに依存します。暗号化アルゴリズムの適用に使われる鍵は、セキュリティ・コンテキストから取得されます。

GSS-API 関数	説明
<code>gss_get_mic()</code>	メッセージから署名を生成し、データが完全なものであり、発信元が信頼できることを保証します。トークン (署名) が戻されます。元のメッセージは、トークンにはカプセル化されません。
<code>gss_unwrap()</code>	トークンの復号と確認を行いその完全性を判別し、必要であればメッセージを復号してメッセージを戻します。
<code>gss_verify_mic()</code>	メッセージとその署名を突き合わせて、送信中にメッセージが破損していないことを保証します。

GSS-API 関数	説明
<code>gss_wrap()</code>	メッセージをトークンにカプセル化し、機密性が指定されていればこれを暗号化します。さらに、メッセージが転送中に破損していないことを保証し、データの発信元の認証を提供するために、署名を入れます。
<code>gss_wrap_size_limit()</code>	使用するネットワークで送信できるトークンの最大サイズが決められている場合に、コンテキストの <code>gss_wrap()</code> を見てメッセージ・サイズの上限を調べます。

次の 4 つの関数は相互にペアの関係になっています。

- `gss_get_mic()` と `gss_verify_mic()`
- `gss_wrap()` と `gss_unwrap()`

アプリケーションは、署名をデータとは切り離して送信したい場合は、`gss_get_mic()` と `gss_verify_mic()` を使用できます。たとえば、PGP メール (Pretty Good Privacy) では、メッセージが最初に送信され、その後に署名が送信されます。署名がチェックされ、その次に結果が表示され、送信中にメッセージが変更されなかったことを示します。

#### 注意

これらの関数は、GSS-API V2 で始めて導入されたものです。以前のリリースでは、関数 `gss_sign()`、`gss_verify()`、`gss_seal()`、`gss_unseal()` を使用していました。

### 8.3.4.1 Quality of Protection

アプリケーションがこれらのメッセージ関数を呼び出すときには、メッセージに適用される完全性または機密性のアルゴリズムを識別する QOP を指定します。Kerberos 5 では、次の 6 つのアルゴリズムが QOP に対して定義されています。

- DES-MAC-MD5
- DES3-MAC-MD5
- MD2.5

- DES-MAC
- DES-CBC
- DES3-CBC

注意

同じ QOP が指定されていると、`gss_get_mic()` と `gss_wrap()` は署名を同じ方法で計算します。

8.3.5 その他のサポート関数

この節では、アプリケーションの保護に使用できる、その他のサポート関数の一覧をまとめます。

GSS-API 関数	説明
<code>gss_add_oid_set_member()</code>	オブジェクト識別子 (OID) をセットに追加します。
<code>gss_create_empty_oid_set()</code>	オブジェクト識別子のないセットを作成します。
<code>gss_display_status()</code>	GSS-API ステータス・コードを表示可能なテキスト形式に変換します。アプリケーションは、いくつかのエラーが示されている複数のステータス・メッセージを受け取るために、この関数を何度も呼び出す必要が生じることもあります。アプリケーションは、 <code>gss_release_buffer()</code> を呼び出すことによってバッファを解放します。この関数は、メジャー・ステータス・コードとマイナー・ステータス・コードの両方を扱います。
<code>gss_indicate_mechs()</code>	ローカル・システムでサポートされているメカニズム識別子を戻します。この構造体は、 <code>gss_release_oid_set()</code> を使ってアプリケーションが解放しなければなりません。



GSS-API 関数	説明
<code>gss_release_buffer()</code>	表示可能な名前, バッファ, トークンの記憶域を解放します。 <code>gss_display_status()</code> , <code>gss_display_name()</code> , <code>gss_export_name()</code> , <code>gss_export_sec_context()</code> , <code>gss_init_sec_context()</code> , <code>gss_accept_sec_context()</code> , <code>csf_gss_acq_user()</code> , <code>gss_wrap()</code> , <code>gss_get_mic()</code> , <code>gss_unwrap()</code> の呼び出し後に使われます。
<code>gss_release_oid_set()</code>	OID セット・オブジェクトの記憶域を解放します。
<code>gss_test_oid_set_member()</code>	オブジェクト識別子がセットのメンバかどうかを判別します。

その他, HP のエクステンション関数があります。

Application Security SDK 関数	説明
<code>cs_oid_cmp()</code>	2 つの OID を比較します。
<code>cs_oid_dup()</code>	OID を複製します。
<code>cs_oid_free()</code>	OID に関連付けられているリソースを解放します。
<code>cs_oid_in_set()</code>	OID が何らかの OID セットに含まれているかどうかを判別します。
<code>cs_oid_set_cmp()</code>	2 つの OID セットを比較します。
<code>cs_oid_set_dup()</code>	OID セットを複製します。
<code>cs_oid_set_free()</code>	OID セットに関連付けられているリソースを解放します。
<code>cs_oid_set_insert()</code>	OID セットに OID を挿入します。 GSS-API v2 準拠のために, 関数 <code>gss_add_oid_set_member()</code> のほうが優先的に使われます。
<code>cs_oid_set_isect()</code>	2 つの OID セットの共通部分となる新しいセットを作成します。
<code>cs_oid_set_union()</code>	2 つの OID セットの集合となる新しいセットを作成します。

Application Security SDK 関数	説明
<code>csf_gss_get_OidAddress()</code>	組み込まれている OID セットのアドレスを取得します。
<code>csf_gss_get_RfcOidSet()</code>	組み込まれている Kerberos OID セットのアドレスを取得します。
<code>csfgss_pPtr()</code>	OID メカニズムまたは OID メカニズム・セットのポインタを取得します。

### 8.3.5.1 OID と OID セット

OID (オブジェクト識別子) と OID セットを理解するには、いくつかの背景情報が役に立ちます。

#### 8.3.5.1.1 OSI

CCITT X.200 に詳細が記述されている OSI は、物理層からユーザ・アプリケーション層までのコンピュータの相互接続を司る、国際的に標準化されたアーキテクチャです。より高い層にあるオブジェクトを抽象的に定義し、より低い層のオブジェクトによって実装することを意図しています。

#### 8.3.5.1.2 ASN.1

OSI の抽象オブジェクトの指定方法を ASN.1 (Abstract Syntax Notation One, CCITT X.208 で定義されています) と呼びます。ASN.1 は、こうした抽象オブジェクトを 1 とゼロの並びとして表現するための一連のルールを定義したものです。ASN.1 は、さまざまなデータ型の定義が可能な、柔軟性のある表記法です。たとえば、整数とビット文字列などの単純な型、セットとシーケンスなどの構造化された型、その他の型に関して定義された複雑な型が可能です。

#### 8.3.5.1.3 オブジェクト識別子

OID (オブジェクト識別子) は、アルゴリズムと属性タイプなど、オブジェクトを識別するために使われる単純なデータ型の 1 つです。OID は、一連の整数コンポーネント、つまり、登録機関によって意味が与えられている値で構成されています。

GSS-API では、OID は名前タイプとメカニズムなどのパラメータを識別するために使われます。下の例で、この仕組みを示します。

### 例 8-1: 文字列の入った構造体を指す定数

---

定数 `GSS_C_NT_USER_NAME` は、次の OID 文字列を含んだ `gss_OID_desc` 構造体を指すように初期設定された `gss_OID` タイプです。

```
{10, (void *)"\x2a\x86\x48\x86\xf7\x12\x01\x02\x01\x01"}
```

これは、次の 10 進値によるオブジェクト識別子に変換できます。

```
{ 1 2 840 113554 1 2 1 1 }
```

これはさらに、次のオブジェクトの階層を表しています。

```
{iso(1) member-body(2) United States(840) mit(113554) infosys(1) gssapi(2)  
generic(1) user_name(1)}
```

---

### 例 8-2: 文字列を指す定数

---

定数 `rfc_krb5_c_OID` は、次の OID 文字列を指す `gss_OID` タイプです。

```
{iso(1) member-body(2) United States(840) mit(113554) infosys(1) gssapi(2)  
krb5v2(3)}
```

---

#### 8.3.5.1.4 OID セット

OID セットは、一連の OID と、その OID 数を表す整数からなる、GSS-API のデータ構造体です。GSS-API 関数の中には、セットを使って多数のさまざまなメカニズムと属性オブジェクトの入出力を行うものもあります。たとえば、`gss_acquire_cred()` を呼び出すと、サポートされているすべてのメカニズムのオブジェクト識別子の入った OID セットが戻されます。`gss_test_oid_set_member()` を使用すると、特定の OID がセット内にあるかどうかを確認できます。

OID セットは、関数 `gss_create_empty_oid_set()` および `gss_add_oid_set_member()` を使って作成することもできます。

#### 8.3.6 V1 準拠関数

Application Security SDK には、GSS-API Version 2 ではサポートされていない関数がいくつかあります。HP は、今回のリリースでは、v1 との相互運用性の目的でこれらの関数を引き続きサポートしています。GSS-API Version 2 を使って記述する新しいアプリケーションでは、v1 の関数ではなく、それに相当する v2 の関数を使用する必要があります。

GSS-API v1 の関数	相当する GSS-API v2 の関数
<code>gss_open()</code>	このアクションは必要なくなりました
<code>gss_close()</code>	このアクションは必要なくなりました
<code>gss_process_context_token()</code>	このアクションは必要なくなりました
<code>gss_seal()</code>	<code>gss_wrap()</code>
<code>gss_sign()</code>	<code>gss_get_mic()</code>
<code>gss_unseal()</code>	<code>gss_unwrap()</code>
<code>gss_verify()</code>	<code>gss_verify_mic()</code>

## 8.4 ベスト・プラクティス

以降の節では、Application Security SDK を使ってカスタム・アプリケーションを作成する際に、セキュリティに関して考慮すべき点と、推奨事項を取り上げます。

### 8.4.1 マルチ・スレッディング

次の関数は、スレッド・セーフではありません。

- `gss_acquire_cred()`
- `gss_add_cred()`
- `csf_gss_acq_user()`
- `csf_gss_inq_user()`
- `csf_gss_release_user()`
- `csf_gss_renew_cred()`

これらの関数はすべて、初回チケットの取得に関するものであるため、アプリケーションのマルチ・スレッドの部分での使用は避けてください。

`gss_init_sec_context()` と `gss_accept_sec_context()` は、どちらもマルチ・スレッド・セーフを保証するために特別な処置がなされています。他の関数はすべて、各スレッドが一意のセキュリティ・コンテキストを使用している限り、スレッド・セーフです。

### 8.4.2 キャッシュ管理

プリンシパル間でのキャッシュの共有は、セキュリティ上危険です。キャッシュを共有するプリンシパルの1つが、そのキャッシュを使っている別のプリンシパルになりすます可能性があるためです。その結果、プリンシパルやプリンシパルを信頼しているサーバが悪用されたり、またはリプレイされたメッセージが受け渡されたりすることが考えられます。

HP では、開始側ごとに一意の信任状キャッシュを使い、受け入れ側ごとに一意の鍵テーブル・エントリを使うことをお勧めしています。また、受け入れ側のアプリケーションごとに、別々の鍵テーブル・ファイルが用意されているのが理想です。

### 8.4.3 暗号化タイプ

Application Security SDK は、DES と DES3 の両方の暗号化タイプをサポートしています。HP は、DES3 暗号化の使用をお勧めしています。DES3 は DES と比べて保護機能が大幅に向上しています。

ただし、DES3 では移植性に制限があります。DES3 暗号化を使っているアプリケーションは、Application Security SDK の実装でしか使用できません。

### 8.4.4 セキュリティ・コンテキストのエクスポート

`gss_export_sec_context()` 関数は、プロセス間でセキュリティ・コンテキストを渡すためのトークンを作成します。このトークンには、コンテキストの秘密鍵が含まれているため、特別に細心の注意を払う必要があります。エクスポートされたセキュリティ・コンテキストは、GSS-API では保護できないため、アプリケーションによって保護しなければなりません。

GSS-API のこの機能を使うアプリケーションは、エクスポートしたセキュリティ・コンテキストをプロセス間で安全な方法で受け渡す必要があります。エクスポートしたセキュリティ・コンテキストは、セキュリティ・コンテキストを作成したホストの範囲内で保持するようにしてください。

### 8.4.5 GSS と Kerberos 5 による鍵管理

秘密鍵は、暗号化アルゴリズムと鍵付きチェックサム・アルゴリズムによって使用され、定期的に変更する必要があります。どのくらいの頻度で変更すべきかは、使用しているアルゴリズムと保護するデータの量に依存します。

頻度を減らす 1 つの方法としては、DES の代わりに DES3 を使うなど、より強力な暗号化アルゴリズムを使うことが挙げられます。

使われている鍵のタイプには、次の 5 つがあります。

- ユーザ・プリンシパルは、プリンシパル・データベースにその秘密鍵 (パスワード) が格納されています。この鍵は、チケット・グランティング・チケット (TGT) を取得するのに使われます。ユーザ・プリンシパルが、同じパスワードを持つ複数の TGT を取得した場合は、有効期限が切れる前にパスワードを変更する必要があります。
- 各 TGT にはセッション鍵が含まれています。この鍵は、サービス・チケットの取得と、チケットの延長に使われます。ユーザ・プリンシパル、またはクライアントや他のサービスの役割を果たすサービス・プリンシパルが、同じ TGT を持つ複数のサービス・チケットを取得した場合は、有効期限が切れる前に、新しい TGT を取得する必要があります。

TGT の存続期間を短くすることもできます。csf\_gss\_acq\_user() に存続期間を指定すると、この関数はこれを最短の値として扱います。存続期間を過ぎた TGT を使わないよう保証するには、CSF\_GSS\_C\_ACQ\_USER\_OPT\_ALWAYS\_FETCH オプションを使います。このオプションは csf\_gss\_acq\_user() に、指定された存続期間を持つ新しい TGT を KDC から取得するように強制します。

- 各サービス・チケットにはセッション鍵が含まれています。この鍵は、セキュリティ・コンテキストの確立に使われます。ユーザ・プリンシパル、または別のサービスに対するクライアントの役割を果たすサービス・プリンシパルが、同じサービス・チケットを使う複数のセキュリティ・コンテキストを確立した場合、サービス・チケットの有効期限が切れる前にサービス・チケットを交換する必要があります。サービス・チケットは一般に、これを取得するために使った TGT と同時に有効期限が切れます。有効期限が切れる前にサービス・チケットを交換する唯一の方法は、新しい TGT を取得することです。これによってサービス・チケットが格納される信任状キャッシュが再初期化されます。
- サービス・プリンシパルは、プリンシパル・データベースとサービス鍵テーブル・ファイルにも秘密鍵が格納されています。セキュリティ・コンテキストを受け取る際には、この鍵を使ってデータを暗号化します。ユーザ・プリンシパルがサービス・プリンシパルのために取得する各サービス・チケットには、この暗号化されたデータが含まれています。

さらに、他のサービスに対するクライアントの役割を果たすサービス・プリンシパルは、その秘密鍵を使って TGT を取得します。サービス・プリンシパルの秘密鍵は、通常は使用頻度が高いため、頻繁に変更する必要があります。これは、HP が、サーバ・アプリケーション間でサービス・プリンシパルの共有をお勧めしない、1 つの理由です。

- 各セキュリティ・コンテキストにはセッション鍵が含まれています。この鍵は、メッセージを保護するときに使われます。セキュリティ・コンテキストに期限はありません。1 つのセキュリティ・コンテキストを使って大量のデータを交換する場合や、1 回の対話が長時間に及ぶ場合は、セキュリティ・コンテキストを定期的に削除して新しいセキュリティ・コンテキストを確立する必要があります。

セキュリティ・コンテキストのセッション鍵以外は信任状キャッシュまたはサービス鍵テーブル・ファイルに格納されるため、信任状キャッシュとサービス鍵テーブル・ファイルは保護しておく必要があります。

#### 8.4.6 マルチ・スレッド関数

次の Application Security SDK 関数は、スレッド・セーフではありません。

- `gss_acquire_cred()`
- `gss_add_cred()`
- `csf_gss_acq_user()`
- `csf_gss_inq_user()`
- `csf_gss_release_user()`
- `csf_gss_renew_cred()`

これらの関数はすべて、初回チケットの取得に関するものであるため、マルチ・スレッド環境での使用は避けてください。プログラムがスレッドを作成する前に信任状が取得されると、その信任状はマルチ・スレッド型の開始側または受け入れ側に渡されることがあります。

`gss_init_sec_context()` と `gss_accept_sec_context()` は、どちらもマルチ・スレッド・セーフを保証するために特別な処置がなされています。Application Security SDK の他の呼び出しはすべて、各スレッドが一意的セキュリティ・コンテキストを使用している限り、スレッド・セーフです。

## 8.4.7 相互認証

相互認証は、セキュリティ・コンテキストを確立する間に、開始側と受け入れ側の両方のプリンシパルに、相手が本人かどうかをチェックすることを義務付けるものです。相互認証が必要かどうかは、関数 `gss_init_sec_context()` の `req_flags` パラメータの `GSS_C_MUTUAL_FLAG` オプションで示されます。

プリンシパルに相互認証の実行を義務付けない場合、メッセージのリプレイが生じる危険があります。相互認証を実行しない場合は、`req_flags` パラメータの `GSS_C_REPLAY_FLAG` を使ってメッセージ・リプレイ検出を有効にしておく必要があります。

## 8.4.8 パスワードの保護

HP は、プリンシパルからのパスワード情報の取得に使われるすべてのダイアログ・ボックスやフォームで、キーボードのエコーをオフにしておくことを推奨しています。これにより、同じ部屋にいる誰かに、コンピュータ画面のパスワードを見られてしまうのを防ぐことができます。

パスワードの格納に使うバッファは、使った後は直ちにゼロに設定する必要があります。

## 8.4.9 リプレイの防止

`gss_init_sec_context()` 関数には、メッセージ・リプレイ検出機能を有効にするかどうかを示すフラグが含まれています (`req_flags` パラメータの `GSS_C_REPLAY_FLAG`)。アプリケーションがコンテキストを確立するときに相互認証を実行しない場合 (`req_flags` パラメータの `GSS_C_MUTUAL_FLAG` フラグが `FALSE` に指定されている場合) メッセージ・リプレイ検出機能を有効にしておくことが重要です。

メッセージ・リプレイ検出は、他のアプリケーションから受信したメッセージを、リプレイ・キャッシュを使って格納することを義務付けます。キャッシュは開始側と受け入れ側の両方にあります。これらのキャッシュはメモリベースであり、サイズは上限まで拡大できます。したがって、リプレイ検出はプロセスのサイズに影響を及ぼすことになります。新しいリプレイ・キャッシュ・エントリは、`gss_unwrap()` と `gss_verify_mic()` の実行中に割り当てられ、アプリケーションにメモリ・リークがあるように見えますが、これは通常の動作です。メッセージ・リプレイ検出を使う



セキュリティ・コンテキストが削除されると、メッセージ・リプレイ・キャッシュ全体も削除されます。

認証のリプレイ検出には、別のタイプのリプレイ・キャッシュが使われます。これは、UNIX の環境変数 `CSFC5RCNAME`、または Windows のレジストリ・エントリ `RepCache` によって制御される、省略時のファイルベースのキャッシュです。セキュリティに影響を及ぼす可能性があるため、このキャッシュがメモリ・ベースのキャッシュになるようには構成できません。このタイプのキャッシュは常にアクティブであり、開始側のアプリケーションによって制御されることはありません。

#### 8.4.10 信任状のリフレッシュ

有効期限のない暗号鍵を含んでいるセキュリティ・パラメータが 2 つあります。1 つはサービス鍵テーブル・エントリであり、もう 1 つはセキュリティ・コンテキストです。HP では、これらの鍵の暗号が解読される可能性を最小限に抑えるために、この 2 つのパラメータを定期的にリフレッシュすることを推奨しています。

#### 8.4.11 リソース管理

多くの関数が、アプリケーションによって使われるセキュリティ情報を格納するために、文字列、バッファ、その他のリソースを作成します。これらのリソースは、使い終わったら、SDK に含まれている適切な関数を呼び出すことによって正しく解放することが重要です。これに失敗すると、セキュリティが損なわれることにつながります。管理を誤ったリソースは、セキュリティ攻撃で取得され、読み取られる可能性があります。

SDK マニュアルの「リファレンス」の項で、取り扱いに注意を要するリソースを作成する関数の内容を解放するための、適切なリソース管理の呼び出しについての詳細が記されています。個々の SDK マニュアルの「リファレンス」の項をよく読み、セキュリティを危険にさらすのを防ぐための正しい手順を判断してください。

#### 8.4.12 サービス鍵テーブル・ファイル

サービス鍵テーブル・ファイルには、それが本人かどうかをチェックするために受け入れ側のアプリケーションが使う信任状が格納されています。このファイルへのアクセスは、ファイルの内容を読み取るユーザ・パーミッショ

ンを与えないことで、厳しく制限する必要があります。トラステッド・サービスだけに、このファイルへのアクセスを許可する必要があります。

サービス鍵テーブル・ファイルの信任状は、有効期限がありません。このため HP では、これらの信任状を定期的に取りフレッシュすることを推奨しています。サービス鍵テーブル・ファイルのエントリを取りフレッシュするには、セキュリティ・サーバによって新しいランダムな鍵を生成し、プリンシパル・データベース管理プログラムを使ってサービス・ホストに抽出します。

### 8.4.13 チケット属性

Kerberos 5 の初回チケットを、各種の属性を付けて要求できます。その属性は、`csf_gss_acq_user()` 関数のフラグによって指定されます。以降の節では、要求できる属性について説明します。

#### 8.4.13.1 転送可能チケット

転送は、あるホストから別のホストへ、TGT を送るメカニズムです。転送された TGT を使って、TGT で指定されている最初のプリンシパルの代わりに、2 番目のホストで新しいサービス・チケットを生成できます。チケットは、トラステッド・プリンシパルだけに転送するようにしてください。

#### 8.4.13.2 事前認証

事前認証は、TGT 要求を使って追加の暗号化データを送信するものです。追加の暗号化データは、Encrypted Timestamp の形式をとります。追加のこの情報により、認証プロセス中にセキュリティをさらに高めることができます。CSF\_GSS\_C\_ACQ\_USER\_OPT\_NOPREAUTH が指定されている場合、事前認証が必要ないことを示しています。

HP では、事前認証を省略することは、セキュリティとは別の理由でそうする必要のある場合を除き、お勧めしていません。このため、どうしても必要な場合以外は、このオプションの使用を避けることをお勧めします。特に Kerberos 5 セキュリティ・メカニズムの場合、初回チケットの取得中に事前認証を行うことで、自分の名前が偽装者によってチケットに書き込まれるのを防ぐことができます。本来ならばこれらのチケットは、チケットに名前が指定されているプリンシパルによってしか使用できません。それだけがチケットを復号するための秘密鍵を所有しているからです。ただし、悪意のある者が何らかの暗号解読機能を持っている場合、事前認証によって、悪意の

ある者が活用できる情報がセキュリティ・サーバから取得されるのを防ぐことができます。また事前認証によってチケット要求の鮮度も保証されます。

#### 8.4.13.3 チケットの存続期間

要求されたチケットの存続期間は、`CSF_GSS_ACQ_USER_OPT_LIFETIME` フラグによって指定されます。セキュリティ・サーバは、要求されたものよりも短い存続期間の初回チケットを戻すことがあります。指定されたレルムのプリンシパル宛てに発行されたチケットの最大存続期間が、特殊なプリンシパルである `krbtgt/REALM@REALM` の設定によって制御されるためです。

Kerberos のバージョンによって課されるチケットの存続期間の制限もあります。Kerberos 4 におけるチケット存続期間の最大値は、21 時間 15 分です。最大存続期間を 1 日とする場合、Kerberos 5 のチケットは 24 時間完全に存続しますが、Kerberos 4 のチケットは 21 時間 15 分で期限切れとなります。

#### 8.4.13.4 チケットの延長期限

`CSF_GSS_C_ACQ_USER_OPT_RENEWABLE` フラグを使うと、チケットを要求するときに延長可能属性を指定できます。ただしセキュリティ・サーバは、要求されたものよりも短い延長期限で初回チケットを戻すことがあります。これは、指定されたレルムのプリンシパル宛てに発行されたチケットの最大延長期限は、特殊なプリンシパルである `krbtgt/REALM@REALM` の設定によって制御されるためです。

延長可能チケットを受け取った後、延長期限は一連の規則によって制御されます。プリンシパルが最初に TGT を要求するときには、存続期間と延長期限は関連付けられません。初回チケットが発行される段階では、許可されている延長可能時間と存続期間は関係がありません。しかし、チケットが延長されると 2 つの時間が関連を持つことになります。

##### 8.4.13.4.1 存続期間と延長の設定に関する一般的な規則

チケットを延長する場合、延長後の存続期間は次の規則に従って決まります。

- チケットは、延長可能チケット属性を持っている場合に限り延長できます。
- 延長可能チケットは、チケットの存続期間中の任意の時点で延長できます。
- 期限の切れたチケットは延長できません。

- 延長可能チケットは、延長期限が切れるまで、複数回延長できます。
- 延長されたチケットは、延長可能チケットの最初の存続期間と同じ存続期間を持ちます。たとえば、チケットの存続期間が 30 分で、延長期限が 50 分だとします。このとき、15 分経った後でチケットが延長されたとします。このような場合、延長可能チケットの存続期間は、やはり 30 分となります。
- 延長期限の残りが、元のチケットの存続期間よりも短いと、チケットが延長されるときに規則が変更されます。延長されたチケットの有効期限は、元のチケットが発行されたときの延長期限を過ぎることはできません。前述の例で、このチケットを延長可能チケットとして最初に取得した後、25 分経った後で、チケットの有効期限が切れる前にチケットを延長したとします。延長後のチケットはこの後、30 分ではなく 25 分間有効になります ( $25 + 25 = 50$  分)。延長可能であり、延長されたチケットが有効となる合計時間は、元のチケットに設定されている最大延長期限以下でなければなりません。
- 延長可能 TGT を指定して取得された代理 (サービス) チケットは、延長可能オプションとその延長期限を継承し、TGT と共に延長されます。これは、サービス・チケットはそれを取得するために使った TGT の属性を (できる限り) 継承するという、全般的な考え方に従ったものです。
- チケットの延長に、パスワードは必要ありません。

## 8.5 移植性のあるアプリケーションの構築

GSS-API 標準は汎用性を持たせて設計されており、GSS-API を組み込んだアプリケーションは、ソース・レベルで互換性があります。しかし、異なるベンダの GSS-API の実装を使用すると、ベンダ間で移植性の問題にぶつかることがあります。

以降の節では、この影響を最小限に抑え、Application Security SDK を使って作成された GSS-API のプラットフォーム間の移植性を高めるために役立つ推奨事項を取り上げます。

### 8.5.1 表示可能な名前の使用と名前の比較

`gss_display_name()` 関数は、内部的な名前 (つまり `gss_import_name()` によって戻された名前) を、表示に適したテキスト形式に変換します。

GSS-API 標準では、表示可能な名前と `gss_display_name()` から戻され

る名前タイプの OID が、`gss_import_name()` への入力に適したものである必要がないため、表示可能な名前のフォーマットは、実装ごとに固有であり、オペレーティング・システムに依存します。表示可能な名前のフォーマットは、実装がクロス・ベンダ、クロス・プラットフォーム、ベンダによるインクリメンタルなリリースなどに関係なく、GSS-API の複数の実装の間で異なっても構いません。このため、表示可能な名前は、GSS-API 標準外部の名前比較関数への入力としては確実なものとはいえません。このような目的には、`gss_export_name()` 関数で作成されるエクスポート名のフォーマットを使用してください。エクスポート名が使われる可能性のある例としては、ACL (アクセス制御リスト) 機能、アカウント管理、診断支援などがあります。

### 8.5.2 メカニズムの指定

`gss_acquire_cred()` や `csf_gss_acq_user()` に OID セットを渡し、`gss_init_sec_context()` と `gss_import_name()` に OID セットを渡すと、技術が変化したときにアプリケーションの移植性は低くなります。たとえば、セキュリティ・メカニズムとして Kerberos を導入した企業が、のちに公開鍵セキュリティ・メカニズムに変更することなどが考えられます。アプリケーションの開発時に Kerberos を指定した OID セットがハード・コードされていると、そのアプリケーションを変更する必要が生じます。さらにメカニズムの OID も変わる可能性があります。

移植性を高めるためには、アプリケーションは、省略時の OID と OID セットの値を関数に渡す必要があります。こうすることで、GSS-API の実装は、使用可能な適切なメカニズムを選択できます。

### 8.5.3 Quality of Protection (QOP) の指定

`gss_wrap()` および `gss_get_mic()` 関数は、メッセージに適用するアルゴリズムを指定する QOP パラメータを受け取ります。QOP パラメータの値は、メカニズムに固有です。省略時の値以外を指定すると (この場合は GSS-API の実装が値を選択します)、メカニズムに対してアプリケーションをハード・コードすることになります。

このため、使用するメカニズムを変更した場合や、現在使用しているメカニズムの機能が変わった場合に、アプリケーションのソース・コードを変更する必要が生じます。

`gss_unwrap()` および `gss_verify_mic()` 関数とその逆の機能を果たす `gss_wrap()` および `gss_get_mic()` 関数は、メッセージの保護に使用する実際の QOP を戻すものと想定されています。アプリケーションがメッセージをエンコードするときに省略時の QOP を指定した場合、逆の関数によって戻される QOP はメカニズムに依存します。

移植性を高めるためには、アプリケーションは、`gss_wrap()` および `gss_get_mic()` 関数を呼び出すときには省略時の QOP 値を指定し、この逆の関数 (`gss_unwrap()` および `gss_verify_mic()`) によって戻される値は使わないようにする必要があります。

省略時の完全性の QOP は次のとおりです。

- DES3 の場合は `CSF_GSS_KRB5_INTEG_C_QOP_DES3_MD5`
- DES の場合は `GSS_KRB5_INTEG_C_QOP_DES_MD5`

機密性の QOP は次のとおりです。

- DES3 の場合は `CSF_GSS_KRB5_CONF_C_QOP_DES3`
- DES の場合は `GSS_KRB5_CONF_C_QOP_DES`

使用される省略時の QOP は、セキュリティ・コンテキストが確立されたときに選択された暗号化タイプに依存します。

---

#### 注意

---

DES3 の QOP には移植がありません。この QOP を指定すると、アプリケーションは Kerberos 5 メカニズム専用になります。

---

## 8.5.4 省略時の名前

いくつかの GSS-API 関数は、省略時の名前を選択するようなパラメータを受け取ります。しかし、GSS-API 標準では、省略時の名前を形成する方法を定義していないため、省略時の名前はメカニズムに依存し、オペレーティング・システムにも依存します。たとえば、ある GSS-API の実装では、環境変数を問い合わせることによって省略時の名前を形成し、GSS-API の別の実装では、ユーザの ID (たとえば UNIX の UID) から形成するなどが考えられます。さらに、間接的な選択基準やメカニズムの慣例に基づいて省略時の名前を形成する GSS-API の実装もあります。たとえば、Application Security SDK の Kerberos メカニズムの実装では、省略時の名前は、

GSS\_C\_INITIATE に使用する信任状の場合は `user@REALM`, GSS\_C\_ACCEPT に使用する信任状の場合は `host/hostname@REALM` となります。

HP では、移植性を高めるために、メカニズムに依存する省略時の名前は使わないことを推奨しています。

## 8.6 クリック・リファレンス

Application Security SDK は、次の表に示すように、アプリケーションの保護に使われる標準の関数と独自の関数からなります。

- `gss` で始まる関数呼び出しは、GSS-API 標準です。
- `cs` または `csf` で始まる関数呼び出しは、HP 固有のエクステンション関数です。

関数	説明
<code>cs_oid_cmp()</code>	2 つの OID を比較します。
<code>cs_oid_dup()</code>	OID を複製します。
<code>cs_oid_free()</code>	OID を解放します。
<code>cs_oid_in_set()</code>	OID が何らかの OID セットに含まれているかどうかを判別します。
<code>cs_oid_set_cmp()</code>	2 つの OID セットを比較します。
<code>cs_oid_set_dup()</code>	OID セットを複製します。
<code>cs_oid_set_free()</code>	OID セットを解放します。
<code>cs_oid_set_insert()</code>	OID セットに OID を挿入します。
<code>cs_oid_set_isect()</code>	既存の 2 つの OID セットの共通部分となる新しい OID セットを作成します。
<code>cs_oid_set_union()</code>	既存の 2 つの OID セットの集合となる新しい OID セットを作成します。
<code>csf_gss_acq_user()</code>	セキュリティ・コンテキストの開始前にユーザを取得します。
<code>csf_gss_get_context_options()</code>	使用されている暗号化のタイプを確認します。
<code>csf_gss_get_OidAddress()</code>	組み込まれている OID のアドレスを戻します。

関数	説明
<code>csf_gss_get_RfcOidSet()</code>	組み込まれている Kerberos の OID セットのアドレスを戻します。
<code>csf_gss_inq_user()</code>	ユーザに関する情報を取得します。
<code>csfgss_pPtr()</code>	OID メカニズムまたは OID メカニズム・セットのポインタを取得します。
<code>csf_gss_release_user()</code>	不要になったユーザを削除します。
<code>csf_gss_renew_cred()</code>	信任状を延長します。
<code>gss_accept_sec_context()</code>	ピア・アプリケーションによって開始されたセキュリティ・コンテキストを受け入れます。
<code>gss_acquire_cred()</code>	格納されている信任状を、アプリケーションでの使用のために取り出します。
<code>gss_add_cred()</code>	信任状を追加的に作成します。
<code>gss_add_oid_set_member()</code>	オブジェクト識別子 (OID) をセットに追加します。
<code>gss_canonicalize_name()</code>	内部名をメカニズム固有の名前 (MIN) に変換します。
<code>gss_close()</code>	v1 との相互運用性のためにサポートされている関数です。
<code>gss_compare_name()</code>	内部名で表現された 2 つの名前を比較します。
<code>gss_context_time()</code>	セキュリティ・コンテキストの有効期限の残り時間を調べます。
<code>gss_create_empty_oid_set()</code>	オブジェクト識別子のないセットを作成します。
<code>gss_delete_sec_context()</code>	セキュリティ・コンテキストを削除します。
<code>gss_display_name()</code>	内部形式の名前をテキストに変換します。
<code>gss_display_status()</code>	GSS-API のステータス・コードをテキストに変換します。
<code>gss_duplicate_name()</code>	既存の内部形式の名前のコピーを作成します。
<code>gss_export_name()</code>	メカニズムに固有の名前をエクスポート形式に変換します。



関数	説明
<code>gss_export_sec_context()</code>	セキュリティ・コンテキストを別のプロセスに転送します。
<code>gss_get_mic()</code>	メッセージの、メッセージ完全性コード (MIC) と呼ばれる署名を計算します。
<code>gss_import_name()</code>	テキスト形式の名前を内部形式に変換します。
<code>gss_import_sec_context()</code>	エクスポートされたセキュリティ・コンテキストをインポートします。
<code>gss_indicate_mechs()</code>	使用可能なセキュリティ・メカニズムを調べます。
<code>gss_init_sec_context()</code>	ピア・アプリケーションとのセキュリティ・コンテキストを開始します。
<code>gss_inquire_context()</code>	セキュリティ・コンテキストに関する情報を取得します。
<code>gss_inquire_cred()</code>	信任状に関する情報を取得します。
<code>gss_inquire_cred_by_mech()</code>	信任状に関するメカニズムごとの情報を取得します。
<code>gss_inquire_mechs_for_name()</code>	指定された名前タイプをサポートするメカニズムの一覧を表示します。
<code>gss_inquire_names_for_mech()</code>	指定されたメカニズムによってサポートされている名前タイプの一覧を表示します。
<code>gss_oid_to_str()</code>	OID を文字列として表示します。
<code>gss_open()</code>	v1 との相互運用性のためにサポートされている関数です。
<code>gss_process_context_token()</code>	ピア・アプリケーションからのセキュリティ・コンテキストのトークンを処理します。v1 との相互運用性のためにサポートされている関数です。
<code>gss_release_buffer()</code>	バッファを削除します。
<code>gss_release_cred()</code>	使用後の信任状を削除します。
<code>gss_release_name()</code>	内部形式の名前を削除します。
<code>gss_release_oid()</code>	OID オブジェクトの記憶域を解放します。
<code>gss_release_oid_set()</code>	オブジェクト識別子のセットを削除します。

関数	説明
<code>gss_seal()</code>	<code>gss_wrap()</code> によって置き換えられています。
<code>gss_sign()</code>	<code>gss_get_mic()</code> によって置き換えられています。
<code>gss_str_to_oid()</code>	文字列から OID を作成します。
<code>gss_test_oid_set_member()</code>	オブジェクト識別子がセットのメンバーかどうかを調べます。
<code>gss_unseal()</code>	<code>gss_unwrap()</code> によって置き換えられています。
<code>gss_unwrap()</code>	メッセージに添付された署名をチェックし、必要であればメッセージの内容を復号します。
<code>gss_verify()</code>	<code>gss_verify_mic()</code> によって置き換えられています。
<code>gss_verify_mic()</code>	受信されたメッセージの署名をチェックします。
<code>gss_wrap()</code>	メッセージに署名を添付し、必要であればメッセージの内容を暗号化します。
<code>gss_wrap_size_limit()</code>	使用するネットワークで送信できるトークンの最大サイズが決められている場合に、コンテキストの <code>gss_wrap()</code> を見てメッセージ・サイズの上限を調べます。

### 8.6.1 リファレンス・ページの表記規則

各関数はリファレンス・ページで、次の項目を使って説明されています。

使用目的

関数の用途の簡単な説明。

構文

使用可能なパラメータと、コマンド行引数の完全なリスト。

入力パラメータは、アプリケーションから Application Security SDK に渡される引数です。

出力パラメータは、Application Security SDK からアプリケーションに渡される引数です。

入力パラメータが省略可能な場合、アプリケーションはそのパラメータに対する省略時の値を渡すことができます。

出力パラメータが省略可能な場合、アプリケーションにとってそのパラメータは必須ではないため、Application Security SDK は値を戻す必要がありません。このことを示す場合は、出力パラメータに `NULL` を使用してください。

---

#### 注意

---

メモリ・リークを避けるために、アプリケーションは常に、戻された値に関連付けられている記憶域を使用後に解放する必要があります。

---

#### パラメータ

関数に渡され、関数から戻される一連の変数。2 進値の場合は、

- 論理 1 は真を表します。
- 論理 0 は偽を表します。

`NULL` に初期設定されているパラメータは、ゼロになることもあります。

#### 説明

関数の定義、使用目的、使用上のヒント。

#### 移植性に関する考慮事項

HP によって保護されたアプリケーションを、別の環境に移植するための考慮事項。

#### リターン値

関数によって戻されるメジャー・ステータス・コードの一覧。

#### 関連項目

関連のある関数呼び出しの一覧。

## 8.7 定数

Application Security SDK に含まれているヘッダ・ファイルでは多数の定数を使用しています。以下に、リファレンス用にヘッダ・ファイルの定数の定義を示します。

コンテキスト・フラグ	定義
GSS_C_DELEG_FLAG	1
GSS_C_MUTUAL_FLAG	2
GSS_C_REPLAY_FLAG	4
GSS_C_SEQUENCE_FLAG	8
GSS_C_CONF_FLAG	16
GSS_C_INTEG_FLAG	32
GSS_C_ANON_FLAG	64
GSS_C_PROT_READY_FLAG	128
GSS_C_TRANS_FLAG	256
CSF_GSS_C_DES_FLAG	268435456
CSF_GSS_C_DES3_FLAG	536870912

信任状の使用	定義
GSS_C_BOTH	0
GSS_C_INITIATE	1
GSS_C_ACCEPT	2

ステータス・コードのタイプ	定義
GSS_C_GSS_CODE	1
GSS_C_MECH_CODE	2

アドレスのタイプ	定義
GSS_C_AF_UNSPEC	0
GSS_C_AF_LOCAL	1
GSS_C_AF_INET	2
GSS_C_AF_IMPLINK	3
GSS_C_AF_PUP	4
GSS_C_AF_CHAOS	5

アドレスのタイプ	定義
GSS_C_AF_NS	6
GSS_C_AF_NBS	7
GSS_C_AF_ECMA	8
GSS_C_AF_DATAKIT	9
GSS_C_AF_CCITT	10
GSS_C_AF_SNA	11
GSS_C_AF_DECnet	12
GSS_C_AF_DLI	13
GSS_C_AF_LAT	14
GSS_C_AF_HYLINK	15
GSS_C_AF_APPLETALK	16
GSS_C_AF_BSC	17
GSS_C_AF_DSS	18
GSS_C_AF_OSI	19
GSS_C_AF_X25	21
GSS_C_AF_NULLADDR	255

各種の NULL 値	定義
GSS_C_NO_NAME	NULL
GSS_C_NO_BUFFER	NULL
GSS_C_NO_OID	NULL
GSS_C_NO_OID_SET	NULL
GSS_C_NO_CONTEXT	NULL
GSS_C_NO_CREDENTIAL	NULL
GSS_C_NO_CHANNEL_BINDINGS	NULL
CSF_GSS_C_NO_USER	NULL
CSF_GSS_C_ACQ_USER_OPT_NONE	NULL

QOP	定義
GSS_C_QOP_DEFAULT	0
GSS_KRB5_CONF_C_QOP_DES	0
GSS_KRB5_INTEG_C_QOP_MD5	1
GSS_KRB5_INTEG_C_QOP_DES_MD5	2

QOP	定義
GSS_KRB5_INTEG_C_QOP_DES_MAC	3
CSF_GSS_KRB5_INTEG_C_QOP_DES3_MD5	5341
CSF_GSS_KRB5_CONF_C_QOP_DES3	5342
ユーザ・オプション	定義
CSF_GSS_C_ACQ_USER_OPT_LIFETIME	1
CSF_GSS_C_ACQ_USER_OPT_RENEWABLE	2
CSF_GSS_C_ACQ_USER_OPT_CCNAME	4
CSF_GSS_C_ACQ_USER_OPT_KTNAME	8
CSF_GSS_C_ACQ_USER_OPT_SVCKEY	16
CSF_GSS_C_ACQ_USER_OPT_ALWAYS_FETCH	256
CSF_GSS_C_ACQ_USER_OPT_FORWARDABLE	32
CSF_GSS_C_ACQ_USER_OPT_PROXIABLE	64
CSF_GSS_C_ACQ_USER_OPT_NOPREAUTH	128
暗号化タイプ	定義
CSF_GSS_C_ENCTYPE_DES_CBC_CRC	1
CSF_GSS_C_ENCTYPE_DES_CBC_MD5	3
CSF_GSS_C_ENCTYPE_DES3_CBC_MD5	5
相互認証のタイプ	定義
CSF_GSS_C_PREAUTH_NONE	0
CSF_GSS_C_PREAUTH_ENC_TIMESTAMP	2
CSF_GSS_C_PREAUTH_ENC_UNIX_TIME	5
チャレンジの状態	定義
CSF_GSS_C_USER_STATE_NULL	0
CSF_GSS_C_USER_STATE_PASSWORD_NOECHO	1
CSF_GSS_C_USER_STATE_CHALLENGE_ECHO	2
CSF_GSS_C_USER_STATE_OTP_ECHO	3
CSF_GSS_C_USER_STATE_PASSWORD_ECHO	4
CSF_GSS_C_USER_STATE_CHALLENGE_NOECHO	5
CSF_GSS_C_USER_STATE_OTP_NOECHO	6

その他	定義
GSS_C_INDEFINITE	0xFFFFFFFF
CSF_GSS_C_PURGE_FLAG	1

## 8.8 データ構造体

Application Security SDK では、関数呼び出しとのデータの受け渡しに、次のデータ構造体を使います。

- `gss_channel_bindings_t` — セキュリティ・コンテキストの通信チャンネルを識別します。
- `gss_buffer_t` — 関数呼び出しの入力データまたは出力データを指定します。
- `csf_gss_opts_t` — ユーザ・コンテキストに対するオプションを指定します。

### 8.8.1 `gss_channel_bindings_t`

`gss_channel_bindings_t` は、セキュリティ・コンテキストの通信チャンネルを識別する情報の入ったデータ構造体のポインタです。「チャンネル・バインディング」で、チャンネル・バインディングがどのように動作するかを説明しています。

`gssapi.h` ヘッダ・ファイルで定義されている構造体は、次のとおりです。

```
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_desc, *gss_channel_bindings_t;
}
```

`initiator_addrtype` および `acceptor_addrtype` フィールドは、`initiator_address` および `acceptor_address` バッファにあるアドレスのタイプを示します。Application Security SDK は、アドレス・フォーマット `GSS_C_AF_INET` をサポートしています。

次の関数呼び出しは、`channel_bindings_t` 構造体を使います。

```
gss_accept_sec_context()
gss_init_sec_context()
```

## 8.8.2 gss\_buffer\_t

`gss_buffer_t` は、関数呼び出しとのデータの受け渡しに使われるデータ構造体のポインタです。実際のデータ構造体であるバッファ記述子は、データの合計バイト数の入った長さフィールドと、実際のデータへのポインタの入った値フィールドからなります。

`gssapi.h` ヘッダ・ファイルで定義されている構造体は、次のとおりです。

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

関数呼び出しによってこのデータ構造体を使って戻されるデータの記憶域は、その関数が割り当てます。アプリケーションは、この領域を使い終わったら、`gss_release_buffer()` を呼び出して解放しなければなりません。使用されていないバッファは、`GSS_C_EMPTY_BUFFER` の値に初期設定されます。

次の関数呼び出しは、オプションの構造体を使います。

```
csf_gss_acq_user()
gss_accept_sec_context()
gss_delete_sec_context()
gss_display_name()
gss_display_status()
gss_export_name()
gss_export_sec_context()
gss_get_mic()
gss_import_name()
gss_import_sec_context()
gss_init_sec_context()
gss_process_context_token()
gss_release_buffer()
gss_unwrap()
gss_verify_mic()
gss_wrap()
```

## 8.8.3 csf\_gss\_opts\_t

`csf_gss_opts_t` は、ユーザ・コンテキストに対するオプションのパラメータを格納するデータ構造体のポインタです。

`ext.h` ヘッダ・ファイルに定義されている 1 個のオプションを格納するための構造体は、次のとおりです。



```
typedef struct csf_gss_mech_opt_desc_struct {
    gss_OID      mechOID; /* Mech to specify option for */
    OM_uint32    id;      /* Identifier of option */
    void *       val;      /* Value associated with option if appropriate */
} csf_gss_mech_opt_desc, *csf_gss_opts_t;
```

オプションごとに 1 つ必要なこれらの構造体の配列を作成する必要があります。配列の最後のレコードの mechOID フィールドは、GSS\_C\_NO\_OID に初期設定します。

次の関数呼び出しは、オプションの構造体を使います。

```
csf_gss_acq_user()
```

## 8.9 リターン値

以下は、アプリケーションのデバッグとトラブルシューティングに役立てるための情報です。

### 8.9.1 定義されているステータス・コード

多くの Application Security SDK 関数は、メジャー・ステータスと、マイナー・ステータスという 2 つのリターン値を持っています。どちらのステータス・コードも、共通の構造体に戻されます。

メジャー・ステータスがゼロの場合、関数は正常に実行されています。メジャー・ステータスがゼロ以外の場合は、Application Security SDK でエラーが発生したか、基盤の Kerberos 5 メカニズムで何か失敗が生じたことを示しています。ゼロ以外のメジャー・ステータスが戻された場合は、マイナー・ステータスを参照して、失敗に関する詳細なエラー・コードを調べてください。

エラーの状態をプログラムのにチェックして復旧するには、Application Security SDK に含まれているヘッダ・ファイルを参照してください。

Application Security SDK は、メジャー・ステータスの値を評価するマクロも提供しています。

### 8.9.2 エラー処理マクロ

リターン値を処理するには、次のマクロを使用します。

名前	説明
<code>GSS_ERROR()</code>	ルーチン・エラーまたは呼び出しエラーが発生したことを示します。
<code>GSS_CALLING_ERROR()</code>	呼び出しエラーが発生したことを示します。
<code>GSS_ROUTINE_ERROR()</code>	ルーチン・エラーが発生したことを示します。
<code>GSS_SUPPLEMENTARY_INFO()</code>	補足的な情報のビットが設定されているかどうかを示します。

### 8.9.2.1 GSS\_ERROR()

#### 構文

`GSS_ERROR(major_status)`

#### パラメータ

`major_status` エラーがあるかどうかを示すメジャー・ステータス。OM\_uint32 型を使います。

#### 説明

`GSS_ERROR()` マクロは、メジャー・ステータスがルーチン・エラーまたは呼び出しエラーを示している場合は、真と評価します。

#### 関連項目

`GSS_CALLING_ERROR()`, `GSS_ROUTINE_ERROR()`

### 8.9.2.2 GSS\_CALLING\_ERROR()

#### 構文

`GSS_CALLING_ERROR(major_status)`

#### パラメータ

`major_status` エラーがあるかどうかを示すメジャー・ステータス。OM\_uint32 型を使います。

説明

GSS\_CALLING\_ERROR( ) マクロは、呼び出しエラーを示す呼び出しビットが設定されていれば、論理 1 (1) と評価します。ビットが設定されていなければ、論理ゼロ (0) と評価します。

呼び出しエラー・ビット

GSS_S_CALL_INACCESSIBLE_READ	01xxxxxx
GSS_S_CALL_INACCESSIBLE_WRITE	02xxxxxx
GSS_S_CALL_BAD_STRUCTURE	03xxxxxx

関連項目

GSS\_ERROR( ) , GSS\_ROUTINE\_ERROR( )

8.9.2.3 GSS\_ROUTINE\_ERROR( )

構文

GSS\_ROUTINE\_ERROR(major\_status)

パラメータ

major\_status                   エラーがあるかどうかを示すメジャー・ステータス。OM\_uint32 型を使います。

説明

GSS\_ROUTINE\_ERROR( ) マクロは、ルーチン・エラーを示す呼び出しビットが設定されていれば、論理 1 (1) と評価します。ビットが設定されていなければ、論理ゼロ (0) と評価します。

呼び出しエラー・ビット

GSS_S_BAD_BINDINGS	xx04xxxx
GSS_S_BAD_STATUS	xx05xxxx
GSS_S_BAD_SIG	xx06xxxx
GSS_S_BAD_MECH	xx01xxxx
GSS_S_BAD_NAME	xx02xxxx
GSS_S_BAD_NAME_TYPE	xx03xxxx
GSS_S_CONTEXT_EXPIRED	xx0Cxxxx
GSS_S_CREDENTIALS_EXPIRED	xx0Bxxxx

GSS_S_DEFECTIVE_CREDENTIAL	xx0Axxxx
GSS_S_DEFECTIVE_TOKEN	xx09xxxx
GSS_S_FAILURE	xx0Dxxxx
GSS_S_NO_CRED	xx07xxxx
GSS_S_NO_CONTEXT	xx08xxxx

#### 関連項目

GSS\_CALLING\_ERROR( ), GSS\_ERROR( )

### 8.9.2.4 GSS\_SUPPLEMENTARY\_INFO( )

#### 構文

```
GSS_SUPPLEMENTARY_ERROR(major_status)
```

#### パラメータ

major\_status                      補足的な情報ビットがあるかどうかを調べるメ  
ジャー・ステータス。OM\_uint32 型を使います。

#### 説明

GSS\_SUPPLEMENTARY\_INFO( ) マクロは、補足的な情報ビットが設定されてい  
れば、論理 1 と評価し、ビットが設定されていなければ、論理 0 と評価し  
ます。特定のビットが設定されているかどうかを調べるには、アプリケー  
ションは、メジャー・ステータスと対象の情報ビットのマクロの AND を  
計算する必要があります。

#### 補足的な情報ビット

GSS_S_CONTINUE_NEEDED	xxxx0001
GSS_S_DUPLICATE_TOKEN	xxxx0002
GSS_S_OLD_TOKEN	xxxx0004
GSS_S_UNSEQ_TOKEN	xxxx0008

#### 関連項目

GSS\_ERROR( )

### 8.9.3 メジャー・ステータス

以下に，Application Security SDK 関数呼び出しによって戻されるメジャー・ステータス・コードを，番号順に示します。

16 進値	リターン・コード	説明
00000000	GSS_S_COMPLETE	関数呼び出しは正常に終了しました。
01xxxxxx	GSS_S_CALL_INACCESSIBLE_READ	必要な入力パラメータを読み取れませんでした。
02xxxxxx	GSS_S_CALL_INACCESSIBLE_WRITE	必要な出力パラメータを読み取れませんでした。
03xxxxxx	GSS_S_CALL_BAD_STRUCTURE	パラメータの形式が正しくありません。
xx01xxxx	GSS_S_BAD_MECH	要求されたメカニズムは無効です。
xx02xxxx	GSS_S_BAD_NAME	無効な名前が指定されました。
xx03xxxx	GSS_S_BAD_NAME_TYPE	サポートされていないタイプの名前が指定されました。
xx04xxxx	GSS_S_BAD_BINDINGS	間違ったチャネル・バインディングが指定されました。
xx05xxxx	GSS_S_BAD_STATUS	間違ったステータス・コードが指定されました。
xx06xxxx	GSS_S_BAD_SIG または GSS_S_BAD_MIC	トークンに無効な MIC が含まれていました。
xx07xxxx	GSS_S_NO_CRED	信任状が指定されていないか，無効であるか，アクセスできません。
xx08xxxx	GSS_S_NO_CONTEXT	コンテキストが確立されませんでした。
xx09xxxx	GSS_S_DEFECTIVE_TOKEN	トークンが無効です。
xx0Axxxx	GSS_S_DEFECTIVE_CREDENTIAL	信任状が無効です。
xx0Bxxxx	GSS_S_CREDENTIALS_EXPIRED	参照されている信任状は期限が切れています。

16 進値	リターン・コード	説明
xx0Cxxxx	GSS_S_CONTEXT_EXPIRED	セキュリティ・コンテキストは期限が切れています。
xx0Dxxxx	GSS_S_FAILURE	何らかの理由で GSS-API レベルでのエラーが発生しました。minor_status パラメータに詳細情報が含まれています。
xx0Exxxx	GSS_S_BAD_QOP	要求された QOP をコンテキストが提供できませんでした。
xx0Fxxxx	GSS_S_UNAUTHORIZED	この操作は、ローカルのセキュリティ・ポリシーによって禁止されています。
xx10xxxx	GSS_S_UNAVAILABLE	この操作またはオプションは無効です。GSS_S_CONTINUE_NEEDED は、補足的な情報ビット・フラグです。まず、GSS_ERROR() を使って、メジャー・ステータスのエラーを検査することによって、GSS_S_CONTINUE_NEEDED を調べます。エラーが示されていない場合は、メジャー・ステータスと GSS_S_CONTINUE_NEEDED のビットごとの AND をとります。
xx11xxxx	GSS_S_DUPLICATE_ELEMENT	要求された信任状要素はすでに存在しています。
xx12xxxx	GSS_S_NAME_NOT_MN	指定された名前はメカニズム名ではありません。

16 進値	リターン・コード	説明
xxxx0001	GSS_S_CONTINUE_NEEDED	セキュリティ・コンテキストの確立を完了するには、ピア・アプリケーションからのトークンが必要です。 gss_init_sec_context() または gss_accept_sec_context() を再度呼び出して、トークンを生成する必要があります。
xxxx0002	GSS_S_DUPLICATE_TOKEN	トークンは以前のトークンの複製でした。
xxxx0004	GSS_S_OLD_TOKEN	トークンの有効期限が切れています。
xxxx0008	GSS_S_UNSEQ_TOKEN	後のトークンが先に処理されました。
xxxx0010	GSS_S_GAP_TOKEN	期待されているメッセージごとのトークンが受信されていません。

#### 8.9.4 マイナー・ステータス

以下に、GSS と Kerberos の組み合わせでのエラー・コードを示します。

16 進値	リターン・コード	説明
087F979E4	GSS_KRB5_S_KG_CTX_BINDINGS	ピア・コンテキスト・バインディングの検査に失敗しました。
087F979E5	GSS_KRB5_S_KG_BAD_AUTHENTICATOR	GSS 認証子が間違っています。
087F979E6	GSS_KRB5_S_KG_NO_MECH	メカニズムがありません。
087F979E7	GSS_KRB5_S_KG_KRB_ERR	トークンは KRB-ERR です。
087F979E8	GSS_KRB5_S_KG_RECV_ADDR	復号には送信者のアドレスが必要です。
087F979E9	GSS_KRB5_S_KG_SENDER_ADDR	暗号化には送信者のアドレスが必要です。

16 進値	リターン・コード	説明
087F979EA	GSS_KRB5_S_KG_BAD_MSG	トークンはプライベート・メッセージまたは安全なメッセージではありません。
087F979EB	GSS_KRB5_S_KG_CTX_INCOMPLETE	完全でないセキュリティ・コンテキストを使おうとしています。
087F979EC	GSS_KRB5_S_KG_BAD_LENGTH	トークンのフィールドの長さが正しくありません。
087F979ED	GSS_KRB5_S_KG_BAD_QOP_USAGE	QOP の使い方が間違っているか、サポートされていません。
087F979EE	GSS_KRB5_S_KG_CONTEXT_ESTABLISHED	コンテキストはすでに完全に確立されています。
087F979EF	GSS_KRB5_S_KG_NO_SUBKEY	認証子にサブキーがありません。
087F979F0	GSS_KRB5_S_KG_TGT_MISSING	信任状キャッシュに TGT がありません。
087F979F1	GSS_KRB5_S_KG_KEYTAB_NOMATCH	鍵テーブルのプリンシパルに要求された名前と一致するものがありません。
087F979F2	GSS_KRB5_S_KG_CCACHE_NOMATCH	信任状キャッシュのプリンシパルが指定された名前と一致していません。
087F979F3	GSS_KRB5_S_G_BAD_KEYGEN	サブキーを生成できません。
087F979F4	GSS_KRB5_S_G_BAD_STATUS	ディスプレイ・ステータス・コードが不明です。
087F979F5	GSS_KRB5_S_G_UNKNOWN_QOP	指定された QOP は不明です。
087F979F6	GSS_KRB5_S_G_BAD_USAGE	信任状の使い方が不明です。
087F979F7	GSS_KRB5_S_G_WRONG_SIZE	バッファのサイズが間違っています。



16 進値	リターン・コード	説明
087F979F8	GSS_KRB5_S_G_BAD_MSG_CTX	メッセージ・コンテキストが無効です。
087F979F9	GSS_KRB5_S_G_BUFFER_ALLOC	gss_buffer_t データを割り当てることができません。
087F979FA	GSS_KRB5_S_G_VALIDATE_FAILED	検査エラーです。
087F979FB	GSS_KRB5_S_G_NOSRVC	名前文字列に SERVICE: プレフィックスがありません。
087F979FC	GSS_KRB5_S_G_UID_LEN	uid_t に対するバッファの長さが間違っています。
087F979FD	GSS_KRB5_S_G_NOUSER	UID がユーザ名を解決していません。
087F979FE	GSS_KRB5_S_G_BAD_UID_NOSUPP	UID はこのオペレーティング・システムではサポートされていません。
087F979FF	GSS_KRB5_S_G_BAD_SERVICE_NAME	名前文字列 SERVICE-NAME に @ がありません。

### 8.9.5 Kerberos 固有のコード

Kerberos 固有のエラー・コードは、マイナー・エラー・コードです。

ステータス・コードのテキスト表現を取得するには、関数 `gss_display_status()` を使います。

戻されるエラー・コードの完全なリストを見るには、`... \include \csf \sts` および `... \include \csfc5 \sts` ディレクトリにあるヘッダ・ファイルを参照してください。



# A

## コーディング例

この付録の例は、トラステッド Tru64 UNIX システムでの、いくつかのルーチンの使用方法を示しています。

### A.1 再認証プログラムのソース・コード (sia-reauth.c)

例 A-1 は、パスワードのチェックを行うプログラムです。

#### 例 A-1: 再認証プログラム

```
#include <sia.h>
#include <siad.h>

#ifdef NOUID
#define NOUID ((uid_t) -1)
#endif

main (argc, argv)
int argc;
char **argv;
{
    int i;
    SIAENTITY *entity = NULL;
    int (*sia_collect)() = sia_collect_trm;
    char uname[32];
    struct passwd *pw;
    uid_t myuid;

    myuid = getluid();
    if (myuid == NOUID)
        myuid = getuid(); /* get ruid */
    pw = getpwuid(myuid);
    if (!pw || !pw->pw_name || !*pw->pw_name) {
        sleep(3); /* slow down attacks */
        (void) fprintf(stderr, "sorry");
        return 1;
    }
    (void) strcpy(uname, pw->pw_name);
    i = sia_ses_init(&entity, argc, argv, NULL, uname, \
    NULL, TRUE, NULL);
    if (i != SIASUCCESS) {
        sleep(3); /* slow down attacks */
        (void) fprintf(stderr, "sorry");
        return 1;
    }
    i = sia_ses_reauthentic(sia_collect, entity);
    if (i != SIASUCCESS) {
        (void) sia_ses_release(&entity);
        sleep(3); /* slow down attacks */
        (void) fprintf(stderr, "sorry");
    }
}
```

### 例 A-1: 再認証プログラム (続き)

---

```
        return 1;
    }
    i = sia_ses_release(&entity);
    if (i != SIASUCCESS) {
        sleep(3); /* slow down attacks */
        (void) fprintf(stderr, "sorry");
        return 1;
    }

    (void) fprintf(stderr, "Ok");

    return 0;
}
```

---

## A.2 スーパユーザ認証プログラムのソース・コード (sia-suauth.c)

例 A-2 は、ユーザ用のデーモン (たとえば、crontab や sendmail) を起動するために、root がそのユーザになることを許可するプログラムです。

### 例 A-2: スーパユーザ認証プログラム

---

```
#include <sia.h>
#include <siad.h>

main (argc, argv)
int argc;
char **argv;
{
    int i;

        i = sia_auth(getuid());
        printf("result is %d", i);

    }

    int sia_auth(uid)
    int uid;
    {

    char uname[32];
        static SIAENTITY *entity=NULL;
        static int oargc = 1;
        static char *oargv[1] = { "siatest" };
        static int (*sia_collect)() = sia_collect_trm;
        struct passwd *pw;

        pw = getpwuid(uid);
        if (!pw) {
            printf("getpwuid failure");
            return 8;
        }
        (void) strcpy(uname, pw->pw_name);
```

### A-2 コーディング例

## 例 A-2: スーパユーザ認証プログラム (続き)

---

```
printf("SIA authentication for uid:%d, uname: %s ", \
                                             uid, uname);
if (sia_ses_init(&entity,oargc,oargv,NULL,uname,NULL, \
                FALSE, NULL) == SIASUCCESS) {
    printf( "sia_ses_init successful");
    entity->authtype = SIA_A_SUAUTH;
if (sia_make_entity_pwd(pw, entity) == SIASUCCESS) {
    printf("sia_make_entity_pwd successful");
}
else {
    printf("sia_make_entity_pwd un-successful");
}

    if ((sia_ses_launch(NULL, entity)) == SIASUCCESS) {
        printf( "sia_ses_launch successful");
    }
    else {
        printf( "sia_ses_launch un-successful");
entity = NULL;
    }
    if ((sia_ses_release(&entity)) == SIASUCCESS) {
        printf( "sia_ses_release successful");
    }
    else {
        printf( "sia_ses_release un-successful");
        return(4);
    }

}
else {
    printf( "sia_ses_init un-successful");
    return(5);
}
printf( "sia **** successful");
return(6);
}
```

---



# B

## 監査可能イベントとエイリアス

この付録では、Tru64 UNIX に提供されている省略時の監査可能イベント (/etc/sec/audit\_events) および省略時の監査イベント・エイリアス (/etc/sec/event\_aliases) のファイルを示します。

### B.1 省略時の監査可能イベント・ファイル

以下に、省略時の /etc/sec/audit\_events ファイルを示します。

```
! Audited system calls:
exit                succeed fail
fork                succeed fail
old open            succeed fail
close               succeed
old creat            succeed fail
link                succeed fail
unlink              succeed fail
execv               succeed fail
chdir               succeed fail
fchdir              succeed fail
mknod               succeed fail
chmod               succeed fail
chown               succeed fail
getfsstat            succeed fail
mount               succeed fail
unmount             succeed fail
setuid              succeed fail
exec_with_loader    succeed fail
ptrace              succeed fail
nrecvmsg             succeed fail
nsendmsg             succeed fail
nrecvfrom            succeed fail
naccept              succeed fail
access               succeed fail
kill                 succeed fail
old stat             succeed fail
setpgid              succeed fail
old lstat            succeed fail
dup                  succeed fail
pipe                 succeed fail
open                 succeed fail
```

setlogin	succeed	fail
acct	succeed	fail
classcntl	succeed	fail
ioctl	succeed	fail
reboot	succeed	fail
revoke	succeed	fail
symlink	succeed	fail
readlink	succeed	fail
execve	succeed	fail
chroot	succeed	fail
old fstat	succeed	fail
vfork	succeed	fail
stat	succeed	fail
lstat	succeed	fail
mmap	succeed	fail
munmap	succeed	fail
mprotect	succeed	fail
old vhangup	succeed	fail
kmodcall	succeed	fail
setgroups	succeed	fail
setpgrp	succeed	fail
table	succeed	fail
sethostname	succeed	fail
dup2	succeed	fail
fstat	succeed	fail
fcntl	succeed	fail
setpriority	succeed	fail
socket	succeed	fail
connect	succeed	fail
accept	succeed	fail
bind	succeed	fail
setsockopt	succeed	fail
recvmsg	succeed	fail
sendmsg	succeed	fail
settimeofday	succeed	fail
fchown	succeed	fail
fchmod	succeed	fail
recvfrom	succeed	fail
setreuid	succeed	fail
setregid	succeed	fail
rename	succeed	fail
truncate	succeed	fail
ftruncate	succeed	fail
setgid	succeed	fail
sendto	succeed	fail
shutdown	succeed	fail
socketpair	succeed	fail
mkdir	succeed	fail
rmdir	succeed	fail
utimes	succeed	fail

## B-2 監査可能イベントとエイリアス



adjtime	succeed	fail
sethostid	succeed	fail
old killpg	succeed	fail
setsid	succeed	fail
pid_unblock	succeed	fail
getdirentries	succeed	fail
statfs	succeed	fail
fstatfs	succeed	fail
setdomainname	succeed	fail
exportfs	succeed	fail
getmnt	succeed	fail
alternate setsid	succeed	fail
swapon	succeed	fail
msgctl	succeed	fail
msgget	succeed	fail
msgrcv	succeed	fail
msgsnd	succeed	fail
semctl	succeed	fail
semget	succeed	fail
semop	succeed	fail
lchown	succeed	fail
shmat	succeed	fail
shmctl	succeed	fail
shmdt	succeed	fail
shmget	succeed	fail
utc_adjtime	succeed	fail
security	succeed	fail
kloadcall	succeed	fail
priocntlset	succeed	fail
sigsendset	succeed	fail
msfs_syscall	succeed	fail
sysinfo	succeed	fail
uadmin	succeed	fail
fuser	succeed	fail
proplist_syscall	succeed	fail
ntp_adjtime	succeed	fail
audcntl	succeed	fail
setsysinfo	succeed	fail
swapctl	succeed	fail
memcntl	succeed	fail
SystemV/unlink	succeed	fail
SystemV/open	succeed	fail
RT/memlk	succeed	fail
RT/memunlk	succeed	fail
RT/psx4_time_drift	succeed	fail
RT/rt_setprio	succeed	fail
! Audited trusted events:		
audit_start	succeed	fail
audit_stop	succeed	fail

audit_setup	succeed	fail
audit_suspend	succeed	fail
audit_log_change	succeed	fail
audit_log_creat	succeed	fail
audit_xmit_fail	succeed	fail
audit_reboot	succeed	fail
audit_log_overwrite	succeed	fail
audit_daemon_exit	succeed	fail
login	succeed	fail
logout	succeed	fail
auth_event	succeed	fail
audgen8	succeed	fail
net_tcp_stray_packet	succeed	fail
net_tcp_syn_timeout	succeed	fail
net_udp_stray_packet	succeed	fail
net_tcp_rejected_conn	succeed	fail
! Audited mach traps:		
lw_wire	succeed	fail
lw_unwire	succeed	fail
init_process	succeed	fail
host_priv_self	succeed	fail
semop_fast	succeed	fail
! Audited mach ipc events:		
task_create	succeed	fail
task_terminate	succeed	fail
task_threads	succeed	fail
thread_terminate	succeed	fail
vm_allocate	succeed	fail
vm_deallocate	succeed	fail
vm_protect	succeed	fail
vm_inherit	succeed	fail
vm_read	succeed	fail
vm_write	succeed	fail
vm_copy	succeed	fail
vm_region	succeed	fail
task_by_unix_pid	succeed	fail
bind_thread_to_cpu	succeed	fail
task_suspend	succeed	fail
task_resume	succeed	fail
task_get_special_port	succeed	fail
task_set_special_port	succeed	fail
thread_create	succeed	fail
thread_suspend	succeed	fail
thread_resume	succeed	fail
thread_set_state	succeed	fail
thread_get_special_port	succeed	fail
thread_set_special_port	succeed	fail
port_allocate	succeed	fail

#### B-4 監査可能イベントとエイリアス

port_deallocate	succeed	fail
port_insert_send	succeed	fail
port_extract_send	succeed	fail
port_insert_receive	succeed	fail
port_extract_receive	succeed	fail
host_processors	succeed	fail
processor_start	succeed	fail
processor_exit	succeed	fail
processor_set_default	succeed	fail
xxx_processor_set_default_priv	succeed	fail
processor_set_tasks	succeed	fail
processor_set_threads	succeed	fail
host_processor_set_priv	succeed	fail
host_processors_name	succeed	fail
host_processor_priv	succeed	fail

## B.2 イベント・エイリアス・ファイルのサンプル

以下に、Tru64 UNIX システムで提供されているサンプルの  
/etc/sec/event\_aliases ファイルを示します。

```
# This is a SAMPLE alias list.  Your alias list should be built to
# satisfy your site's requirements.

obj_creat: "old open" "old creat" link mknod open symlink mkdir SystemV/open

obj_delete: unlink truncate ftruncate SystemV/unlink rmdir

exec: execv exec_with_loader execve

obj_access: access "old stat" "old lstat" "old open" open statfs fstatfs \
    readlink "old fstat" stat lstat fstat close:1:0 dup dup2 fcntl \
    "old creat" mmap munmap mprotect memcntl SystemV/open

obj_modify: chmod chown fchown fchmod lchown utimes rename

ipc: recvmsg nrecvmsg recvfrom nrecvfrom sendmsg nsendmsg sendto accept \
    naccept connect socket bind shutdown socketpair pipe sysV_ipc \
    kill "old killpg" setsockopt sigsendset net_tcp_rejected_conn \
    net_udp_stray_packet

sysV_ipc: msgctl msgget msgrcv msgsnd shmat shmctl shmdt shmget semctl \
    semget semop

proc: exit fork chdir fchdir setuid ptrace setpgid setlogin chroot vfork \
    setgroups setpgrp setpriority setreuid setregid setgid audcntl \
    RT/rt_setprio setsid "alternate setsid" priocntlset

system: getfsstat mount unmount acct reboot table sethostname settimeofday \
    adjtime sethostid setdomainname exportfs getmnt swapon utc_adjtime \
    audcntl setsysinfo kloadcall getdirentries revoke "old vhangup" kmodcall \
    security sysinfo uadmin swapctl

misc: ioctl msfs_syscall fuser

trusted_event: login logout auth_event audgen8
```

```

all: obj_creat obj_delete exec obj_access obj_modify ipc proc system misc \
    trusted_event

#+++++

# adjtime is being called once a sec?

profile_audit: audit_start:1:1 audit_stop:1:1 audit_setup:1:1 audit_log_creat:1:
1 audit_xmit_fail:1:1 \
audit_reboot:1:1 audit_log_overwrite:1:1 audit_daemon_exit:1:1 \
settimeofday:1:1 ntp_adjtime:1:1 utc_adjtime:1:1

profile_net: connect:1:1 accept:1:1 bind:1:1 net_udp_stray_packet:1:1 net_tcp_re
jected_conn:1:1

profile_netmon: net_tcp_rejected_conn:1:1 net_tcp_syn_timeout:1:1 net_tcp_stray_
packet:1:1 net_udp_stray_packet:1:1

profile_auth: login:1:1 logout:1:1 auth_event:1:1

profile_filesys: mount:1:1 unmount:1:1

profile_creat: "old creat" link mknod symlink mkdir

profile_proc: setuid setgid setlogin chroot \
    setsid "alternate setsid"

# Definition of catagories
#=====
# Desktop:
# Provides suggested minimal auditing configuration for a single user system. C
onfiguration provides
# monitoring of tusted audit events, no monitoring of files, or network related
events.
# -----
# This alias assumes:
#   - Local access is primarily interactive login, generally limited to one user
#   at a time, activity tracked and controlled by the system.
#   - Individual accountability is primarily maintained by the system.
#   - User related file area access is only limited by file owner choice.
# Browsing is unrestricted.
#   - System related file areas are mostly readonly. Browsing is unrestricted.
#   - Login uid is converted to username.
#   - Access to the network is monitored.
#   - Access to controlled files are unmonitored.
Desktop: \
profile_audit \
profile_auth

# Servers:
# Provides suggested auditing configuration for a system which is used as a ser
ver for networked based
# applications (such as databases, web server, etc.). Configuration provides mon
itoring of trusted
# events, system files, network related files, and network related events.
# -----
# This alias assumes:
#   -Network access is restricted to application (mail, db server, firewall,
#   etc.) controlled access through network mechanisms (tcp/ip reserved port,
#   DECnet objects, etc.) with the application being responsible for tracking
#   activity. Interactive access is strictly controlled by the system, activity
#   is tracked by the system. Application primarily handle access control,
#   system control is secondary.
#   - Local access logins are strictly controlled, activity is tracked by the
#   system.

```

## B-6 監査可能イベントとエイリアス

```

# - Individual accountability is primarily maintained by the applications.
# - User related file area access is strictly limited to application related
# files. Browsing is controlled.
# - system related file areas are at most readonly for user application related
# functions. Browsing is controlled by applications.
# - Login uid is converted to username.
# - Access to the network is monitored.
# - Access to controlled files are monitored.
Server: \
profile_audit \
profile_auth \
profile_net \
profile_filesys \
profile_proc \
profile_creat obj_delete obj_modify

# Timesharing:
# Provides suggested minimal auditing configuration for a system which is used
# to support multiple
# interactive users. Configuration provides monitoring of trusted events, no mon
# itoring of system
# files, or network related events or files.
# -----
# This alias assumes:
# - Local access is primarily interactive login, activity is tracked and
# controlled by the system.
# - Individual accountability is primarily maintained by the system.
# - Interactive logins are generally unrestricted.
# - User related file area access is only limited by file owner choice.
# Browsing is unrestricted.
# - System related file areas are mostly readonly. Browsing is unrestricted.
# - Login uid is converted to username.
# - Access to the network is unmonitored.
# - Access to controlled files is unmonitored.
Timesharing: \
profile_audit \
profile_auth

# Timesharing_extended_audit:
# Provides suggested auditing configuration for a system which is used to suppo
# rt multiple interactive
# users. Configuration provides monitoring of trusted events, system files, and
# no monitoring of network
# related events or files.
# -----
# This alias assumes:
# - Local access is primarily interactive login, activity is tracked and
# controlled by the system.
# - Individual accountability is primarily maintained by the system.
# - Interactive logins are generally unrestricted. # - User related file area
# access is only limited by file owner choice.
# Browsing is unrestricted.
# - System related file areas are mostly readonly. Browsing is unrestricted.
# - Access to the network is monitored.
# - Access to controlled files is monitored.
Timesharing_extended_audit: \
profile_audit \
profile_auth \
profile_filesys \
profile_proc \
profile_creat obj_delete obj_modify

# Networked_system:
# Provides suggested auditing configuration for a system which has networking e

```

```

nabled. Should be used in
# conjunction with Desktop, Timesharing, or Timesharing_extended_audit templates.
Configuration provides
# monitoring of trusted events, network related files and network related events
.
# -----
# This alias assumes:
# - Network access is through application (mail, pinter, etc.) controlled
# network mechanisms (tcp/ip reserved port, DECnet objects, etc.) which are
# responsible tracking activity and controlling access, and Interactive login
# with the system tracking activity and controlling access.
# - Access to the network is monitored.
# - Access to controlled files is monitored.
Networked_system: \
profile_audit \
profile_auth \
profile_net \
profile_creat obj_delete obj_modify

# NIS_server:
# Provides suggested auditing configuration for a system used as a NIS server.
Should be used in
# conjunction with Desktop, Timesharing, or Timesharing_extended_audit templates.
Configuration provides
# monitoring of trusted events, NIS related files and network related events.
# -----
# This alias assumes:
# - Network access is through application (mail, pinter, etc.) controlled
# network mechanisms (tcp/ip reserved port, DECnet objects, etc.) which are
# responsible tracking activity and controlling access, and Interactive login
# with the system tracking activity and controlling access. NIS is enabled.
# - Access to the network is monitored.
# - Access to controlled files is monitored.
NIS_server: \
profile_audit \
profile_net \
profile_creat obj_delete obj_modify

```

# C

## GSS-API チュートリアル

この付録では、GSS-API を使用してアプリケーションをセキュアにする方法を、C 言語によるサンプル・コードを用いて説明します。Application Security SDK に含まれているサンプル・プログラムについても説明します。次の内容について説明しています。

- セキュリティの基礎
- はじめに
- 基本的な GSS-API 関数の使用
- 手順 1：名前の取得
- 手順 2：信任状の取得
- 手順 3：セキュリティ・コンテキストの確立
- 手順 4：メッセージの交換
- 手順 5：セキュリティ・コンテキストの終了
- 高度な概念
- GSS-API 関数のステータス・コード
- サンプル・プログラム

### C.1 セキュリティの基礎

Kerberos を使ったネットワーク認証に慣れていない方は、まずこの章をお読みください。ここでは以下の内容を取り上げます。

- 基本概念
- Kerberos セキュリティ・モデル

#### C.1.1 基本概念

最初に次の基本概念について説明します。

- 識別 — あなたは誰なのか，という情報です。セキュアなシステムは，システムの ID 情報，および個人を一意に特定できる特徴 (スマート・カードやパスワード) などに基づいてこの質問に答えます。
- 認証 — あなたは本人であることを証明できるか，という情報です。認証プロセスでは，セキュリティ・サーバなどの第三者を通じて，あなただと主張している人物が本当にあなたかどうかをチェックします。認証は，メッセージの発信元を証明するためにも使われます。
- 承認 — このシステムであなたに許可されている操作は何か，という情報です。承認は認証に依存します。あなたが本人であることをシステムが認識すると，あなたには ID 情報に基づいてアクセス許可が割り当てられます。
- 機密性 — メッセージが無許可の開示から保護されているかどうかを示します。暗号化は，機密性を実装する手段です。情報の解読方法を知っている人だけが，これにアクセスできます。暗号化と復号は，通常は何らかの鍵 (秘密) とアルゴリズム (公開) によって実現されます。
- 完全性 — メッセージが送信中に破損していないかどうかを示します。完全性は通常，送信の前後に，メッセージに基づいて計算されたチェックサムを比較することによって確認されます。メッセージ・ストリームの完全性は，リプレイ検出，メッセージの順序付けなどの機能によっても提供されます。

## C.1.2 Kerberos セキュリティ・モデル

Kerberos は，マサチューセッツ工科大学で開発された，双方向の第三者による認証および鍵配布システムです。

### C.1.2.1 定義

クライアント — ユーザに代わってネットワーク・サービスを利用するプログラムまたはプロセス。場合によっては，サーバ自身が他のサーバのクライアントになることがあります (たとえば，プリント・サーバは，ファイル・サーバのクライアントです)。

ユーザ — クライアント・プログラムを使う人。

アプリケーション・サーバおよびサービス — 分散型アプリケーションの一方からのサービスを提供するホスト・システム。ユーザはクライアン



ト・プログラムを実行して、アプリケーション・サーバ上で実行しているサービスにアクセスします。

プリンシパル — HP のセキュリティ・システムを使ってエンティティを識別する手段。つまり、セキュリティ・サーバを使って秘密鍵を保存したユーザ、クライアント、サービス、アプリケーション、ホスト・システムなど。

**Key Distribution Center (KDC)** — プリンシパル・データベース、認証サーバ、チケット・グランティング・サービスを保持するセキュリティ・サーバ。これらの KDC コンポーネントは、プリンシパルに信任状を提供し、プリンシパルはこの信任状によって本人であることを証明し、メッセージを交換できるようになります。

- プリンシパル・データベースはネットワーク認証で使われるすべてのプリンシパルに対する秘密鍵を保持しています。
- 認証サービスはプリンシパルの識別情報をチェックし、チケット・グランティング・チケット (TGT) を発行します。
- チケット・グランティング・サービスは TGT と認証子の検査を終えた後でサービス・チケットを配布します。

プリンシパル・データベース — Kerberos 認証で使われる共有の秘密 (つまり秘密鍵) を保持するデータベース (通常は KDC と同じマシンに置かれます)。各ユーザはこのデータベースにプリンシパルを持っています。

レルム — 1 つ以上の KDC によってサービスが提供されているネットワーク上の、一連のプリンシパルを表す名前。1 つのレルム内のプリンシパル名はすべて、一意でなければなりません。1 つのレルム内では、管理ポリシーはすべてのプリンシパルに対して同じになります。

サービス — サーバ上で実行し、ネットワーク上のクライアントから利用可能なアプリケーション。

鍵 — 暗号化と復号の処理の中で使われる、パスワードなどの秘密鍵。

チケット — あるプリンシパルを別のプリンシパルに対して認証するための信任状。KDC だけが、Kerberos チケットを発行できます。

#### C.1.2.2 概念と処理手順

以降の節では、Kerberos の概念と処理手順について説明します。

#### C.1.2.2.1 共有秘密

相互に安全なやり取りを望んでいる 2 つのプリンシパルは、まず、秘密鍵を共有する必要があります。セッション鍵と呼ばれるこの秘密鍵は、ランダムに生成され、やり取りをしようとしている 2 つのプリンシパルだけに与えられます。セッション鍵は、認証と暗号化のために両方のプリンシパルによって使われます。

#### C.1.2.2.2 信用できる第三者による調停

各プリンシパルは秘密鍵を持っています。秘密鍵は、プリンシパルが変更しない限り、変更されることはありません。一方セッション鍵はランダムに生成され、1 回のセッションで 2 つのプリンシパル間で必要な間だけ存在します。

サービスはクライアントを信用せず、クライアントもまたサービスを信用しません。クライアントとサービスは、セッション鍵の生成と配布を行う Key Distribution Center だけを信用します。

#### C.1.2.2.3 Kerberos ネットワーク

Kerberos ネットワークは、レルムと呼ばれるセキュリティ・ドメインに分かれています。各レルムは専用の認証サーバ (KDC) を持っており、専用のセキュリティ・ポリシーを実装しています。これにより Kerberos を実装する組織は、組織内の情報の種類に応じてさまざまなレベルのセキュリティを持つことができます。

レルムは他のレルムからの認証を受け入れることができます。あるいは情報セキュリティ・ポリシーが再認証を要求する場合は、再認証なしでは受け入れないこともできます。これはレルム間認証と呼ばれます。

レルムは階層構造です。つまり、各レルムは子レルムを持つことや、親レルムを持つことがあります。このような構造により、直接的なやり取りのないレルム同士が認証情報を共有できるようになっています。

#### C.1.2.2.4 認証までの 3 つの段階

Kerberos の認証処理には、次の 3 つの段階があります。

1. 初回信任状 クライアントが、パスワードを使って、他のサービスへのアクセス要求の中で使う初回信任状 (チケット・グランティング・チケット, TGT) を取得する。

### C-4 GSS-API チュートリアル

初回信任状を取得するときに、クライアントと KDC は、認証サービス・メッセージを交換します。

2. クライアントが特定のサービスに対する認証を要求する。

サービス・チケットを取得したら、クライアントと KDC が、チケット・グランティング・サービス・メッセージを交換します。

3. クライアントが、サービスに対してサービス・チケットを提示する。

これを行うために、クライアントと要求されたサービスがアプリケーション・メッセージを交換します。

#### C.1.2.2.5 認証サービス・メッセージの交換

認証サービス(Authentication Service, AS)の交換では、チケット・グランティング・サービス(Ticket-Granting Service, TGS)で使う初回信任状とセッション鍵(共有秘密)がクライアントに提供されます。これらの信任状チケット・グランティング・サービスに提示して、クライアントが本人であることを特定のサービスに対して証明する信任状(つまり、サービス・チケット)を要求できます。

このメッセージ交換は、クライアントによって生成された要求と、KDC 認証サービスによって生成された応答からなります。この交換の最後に、クライアントはチケット・グランティング・サービスに対して自分自身を認証できる初回信任状(チケット・グランティング・チケット)を取得します。

認証サービスは、クライアントから要求メッセージを受け取るときに、そのクライアントを認識しているかどうかをチェックします。認識していれば、ランダムなセッション鍵(これがクライアントとチケット・グランティング・サービスの間の共有秘密になります)と TGT(クライアントが、チケット・グランティング・サービスに対して自身を認証するために使われます)を生成します。

認証サービスは、応答メッセージの一部として、セッション鍵と TGT の両方を送信します。認証サービスにはクライアントが認証されたものかどうかはわからないため、クライアントの秘密鍵を使って応答メッセージの一部を暗号化します。認証されているクライアントだけがこの部分を復号できるとわかっているからです。

#### C.1.2.2.6 チケット・グランティング・サービス・メッセージの交換

1 つのチケットは、1 回のサービスに対してのみ有効です。これには、チケット・グランティング・サービスのサービスを使うためのチケットである TGT も含まれます。このため、クライアントが使いたいサービスごとに、異なるサービス・チケットを取得する必要があります。クライアントは、KDC のチケット・グランティング・サービスにメッセージを送ることによって、チケット・グランティング・サービスからサービス・チケットを取得します。本人であることを証明するために TGT 使われるため、ここではユーザ名とパスワードのプロンプトは必要ありません。

クライアントは、(TGT も含む) チケットをサービスに提示するときに認証子も提示します。認証子には、チケットのセッション鍵を使って暗号化されたクライアント名が格納されています。このような方法で、サービスがチケットのセッション鍵を使って認証子を復号でき、クライアントの名前がチケット内の名前と一致していれば、クライアントは認証されたことを保証できます。サービスはクライアントがセッション鍵を知っていると認定し、承認されていない信任状のリプレイを検出する手助けをします。認証子は 1 回のみ使用できます。

チケット要求メッセージを受け取った後、チケット・グランティング・サービスは TGT を復号します。その後、認証子を復号するためにセッション鍵が使われます。クライアント名が認証子と TGT の両方で一致すると、クライアントは、チケット・グランティング・サービスに対して自身を認証することに成功し、サービス・チケットが付与されます。

サービス・チケットは、サービスの秘密鍵を使って暗号化された応答メッセージに入れられて、クライアントに送信されます。メッセージのうち暗号化された部分は、クライアントの秘密鍵ではなく TGT セッション鍵を使って暗号化されています。このため、ユーザはパスワードをもう一度入力する必要はありません。

応答メッセージを受け取った後、クライアントは暗号化された部分を復号します。その後、クライアントはサービス・チケットとそれに対応するセッション鍵を、信任状キャッシュに格納します。

#### C.1.2.2.7 アプリケーション間のメッセージの交換

クライアントは、サービスに対するサービス・チケットを取得した後、サービスに対して自身を認証し、サービスとの間で安全にメッセージを交換でき

るようになります。クライアントは、サービス・チケットを認証子と共に検証のためにサービスに送信します。

#### C.1.2.3 信任状の属性

クライアントは、KDC によって発行された信任状に次のような特徴を付加することを要求できます。

- LIFETIME — 信任状は、特定の期間を対象として発行されます。この期間は通常はローカルのセキュリティ・ポリシーによって決定されます。
- RENEWABLE — 信任状の有効期限が切れる前に、プリンシパル名とパスワードの入力なしでクライアントがこの期限を延長できる属性です。
- POSTDATED — クライアントは、後で使用するために POSTDATED チケットを要求できます。たとえば、このチケットはバッチ・ジョブを処理するためにアプリケーションによって使われることなどが考えられます。
- FORWARDABLE — この信任状は、サービスがクライアントに代わって、クライアントのチケットを使って何らかの操作を実行できるようにする必要がある場合に使われます。

## C.2 はじめに

Application Security SDK を使って分散型アプリケーションを保護する前に、考えなければならない最初の決定事項は、次のとおりです。

1. このアプリケーションを開始する前に、アプリケーション・ユーザ用の初回信任状 (TGT) は用意されているか。

用意しないのであれば、アプリケーションは `csf_gss_acq_user()` を呼び出すことによってユーザ・コンテキストを確立する必要があります。この関数は、アプリケーション・ユーザのために、初回信任状 (TGT) をセキュリティ・サーバから取得します。これらの信任状は、以降の処理の間、開始側のアプリケーションによって使われます。本書ではこれらを、しばしば「開始側の信任状」と呼びます。

DES3 暗号化または公開鍵の信任状が使われている場合、アプリケーションはこれらの属性を `csf_gss_acq_user()` に渡す必要があります。

2. アプリケーションには相互認証が必要か。

相互認証は、開始側のアプリケーションが、受け入れ側のアプリケーションの識別情報を検証する必要があるときに推奨されます。たとえば受け入れ側のアプリケーションにクレジット・カード番号などの機密性の高いデータを送信する場合などです。

相互認証が必要であれば、開始側のアプリケーションは、`gss_init_sec_context()` を呼び出して、コンテキスト確立時に相互認証を要求する必要があります。

3. (もしあれば) どのメッセージをアプリケーションとそのやり取り相手の間で保護する必要があるか。

中には、アプリケーションが必要としているサービスが、認証だけの場合もあります。

4. メッセージを保護する必要がある場合は、どの種類の保護が必要か。

保護のタイプは、機密性と完全性という 2 つの基本的なカテゴリに分類されます。

- 機密性 (またはプライバシー) は、メッセージが侵入者に読まれるのを防ぎます。
- 完全性は、メッセージが侵入者によって改変されるのを防ぎます。たとえば、金融取引のときに完全性サービスを使って、金額を書き換えられたり、口座番号を変更されたりするのを防ぎます。

5. メッセージの保護のためにどのような Quality of Protection (QOP) が必要か。

最も低いレベル (DES 暗号化) は、高速ですが安全性は低くなります。最も高いレベル (DES3 暗号化) は、時間がかかりますが安全性は高くなります。アプリケーションが DES3 暗号化を使用する場合は、いくつかの前提条件を満たす必要があります。

### C.3 基本的な GSS-API 関数の使用

GSS-API においては、セキュリティ・コンテキストは開始されるか受け入れられるかのどちらかになります。アプリケーションの役割によって、それがセキュリティ・コンテキストを開始するのか、受け入れるのかが決まります。クライアント/サーバ・アプリケーションでは、通常はクライアントはセキュリティ・コンテキストを開始し、サーバはセキュリティ・コン

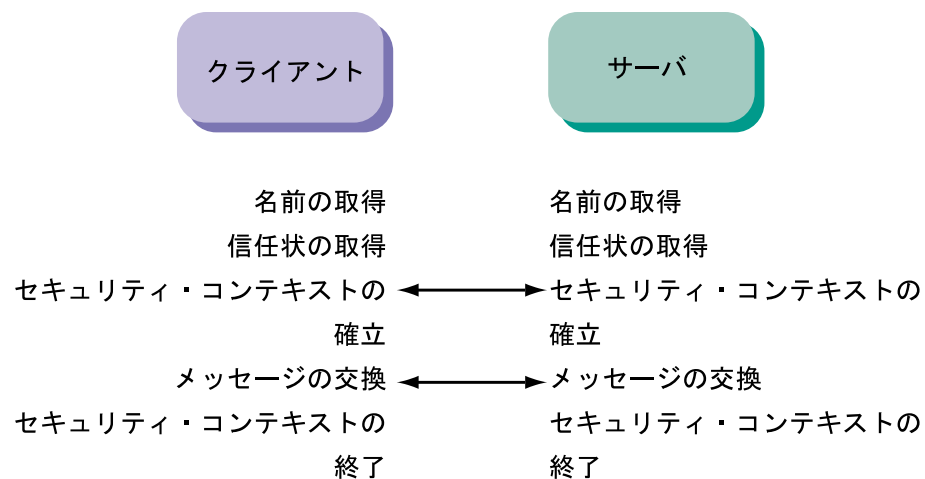
テキストを受け入れます。サーバが別のセキュア・サービスのクライアントになることもあります。

GSS-API 関数を使って開始側と受け入れ側の間でセキュリティを実装する手順は、大きく分けて次の 5 つからなります。

1. 名前の取得
2. 信任状の取得
3. セキュリティ・コンテキストの確立
4. メッセージの交換
5. セキュリティ・コンテキストの終了

開始側と受け入れ側の両方のアプリケーションが、この手順を同じ順番で実行する必要があります。ただし、すべての手順が必須というわけではありません。たとえば、信任状の取得は、省略時の信任状を使う場合は省略できます。

両方のアプリケーションが、下に示す順番でこれらの手順を実行する必要があります。たとえば、アプリケーション間でコンテキストを確立するには、アプリケーションの双方が信任状を取得しなければなりません。



ZK-1827U-AI

ここで取り上げる基本的な GSS 関数の説明では、開始側のアプリケーション (開始側) はユーザ、受け入れ側のアプリケーション (受け入れ側) はサービスであると仮定します。

以降の説明では、両方のアプリケーションはすでに、セキュリティ・サーバから初回信任状 (つまり TGT)、または秘密鍵を取得しているものとします。実行時に初回信任状を取得する場合は、HP の GSS-API 拡張である `csf_gss_acq_user()` を使ってユーザ・コンテキストを確立する必要があります。

以降の節の各手順で説明されている関数は、使用可能な GSS-API 関数のサブセットです。これらは、Application Security SDK でアプリケーションを保護するために最低限必要な呼び出しです。

## C.4 手順 1： 名前の取得

アプリケーションを保護するためには、この処理にかかわる対象の名前を取得する必要があります。この手順では、開始側のユーザ名と、受け入れ側のサービス名をインポートする方法を説明します。これらの名前は、信任状の取得とセキュリティ・コンテキストの確立に使われます。

`gss_import_name()` 関数を使って、名前を他の GSS-API 関数が使用できるような内部形式に変換します。さまざまな形式、つまり名前タイプを変換のタイプとして指定できます。

内部形式による名前は、表示しても判別できませんが、`gss_display_name()` を使って人間が判読できる形式に変換できます。この関数は、省略時の信任状を取得した場合に、既存の信任状に対応するプリンシパルを表示するのに便利です。

開始側と受け入れ側のどちらかがプリンシパル名 (ユーザ名またはサービス名) をインポートし、表示するときに、同じ処理が発生します。「GSS-API 名前管理関数」で、他の名前関数を使って実行できることについて説明されています。

### 注意

開始側からの `gss_import_name()` の呼び出しは省略可能です。`gss_import_name()` が開始側によって呼び出されなければ、省略時の信任状が要求された場合、`gss_acquire_cred()` または `gss_init_sec_context()` が呼び出されるときに、省略時のプリンシパル名が信任状キャッシュから取り出されます。ただし、`gss_init_sec_context()` の呼び出しによって受け入



れ側の名前が要求されるため、開始側でその名前をインポートする必要があります。

受け入れ側からの `gss_import_name()` の呼び出しもまた省略可能です。`gss_import_name()` が開始側によって呼び出されなければ、省略時のサービス・プリンシパル名 (`host/hostname@REALM`) が、`gss_acquire_cred()` または `gss_accept_sec_context()` の呼び出し時に作成されます。

`gss_import_name()` 関数には次のパラメータがあります。

---

#### 入力パラメータ

---

<code>input_name_buffer</code>	変換するテキスト名。
<code>input_name_type</code>	名前のタイプ — 公開鍵, Kerberos, 汎用名タイプがサポートされています。

---

#### 出力パラメータ

---

<code>minor_status</code>	Kerberos 5 エラー・コード。
<code>output_name</code>	内部形式による名前が戻されます。

---

サンプル・プログラムから抜粋した下記のコードは、開始側の名前をインポートして、人間が判読可能な形式に変換する方法を示しています。名前タイプには、Kerberos 5 の省略時の値、つまりここでは `GSS_C_NT_USER_NAME` が指定されています。これはユーザ・プリンシパルに対してよく使われる名前タイプです。この名前タイプは、ユーザ名 `user` を `user@LOCAL_REALM` という形式に変換します。

```
/******  
  
char          szUserName[]      = "walth@APPSEC.CYBERSAFE.COM";  
OM_uint32     gMaj, gMin       = 0;  
gss_buffer_desc input_name      = GSS_C_EMPTY_BUFFER;  
gss_buffer_desc display_name    = GSS_C_EMPTY_BUFFER;  
gss_name_t     principal_name   = GSS_C_NO_NAME;  
gss_OID        display_name_type = GSS_C_NO_OID;  
  
/* Copy the principal name into the GSS buffer structure */  
input_name.length = strlen( szUserName );  
input_name.value  = strdup( szUserName );  
  
/* Convert the user principal name into GSS internal form */  
gMaj = gss_import_name( &gMin,  
                        &input_name,  
                        GSS_C_NT_USER_NAME,  
                        &principal_name );
```

```

/* Convert the imported name to human-readable form */
gMaj = gss_display_name( &gMin,
                        principal_name,
                        &display_name,
                        &display_name_type );

/*****

```

サンプル・プログラムから抜粋した下記のコードは、受け入れ側の名前をインポートして、人間が判読可能な形式に変換する方法を示しています。名前タイプには、Kerberos 5 の省略時の値、つまりここでは GSS\_KRB5\_NT\_HOSTBASED\_SERVICE\_NAME が指定されています。これは無人運用ホストに対してよく使われる名前タイプです。この名前タイプは、Kerberos のホスト・サービス・プリンシパル *service@host* を *service/fqdn@REALM* の形式に変換します。たとえば、*ftp@fuji* は *ftp/fuji.company.com@COMPANY.COM* に変換されます。

```

/*****

char          szHostName[]      = "host@dev007.cybersafe.com";
OM_uint32     gMaj, gMin       = 0;
gss_buffer_desc input_name     = GSS_C_EMPTY_BUFFER;
gss_buffer_desc display_name   = GSS_C_EMPTY_BUFFER;
gss_name_t     principal_name  = GSS_C_NO_NAME;
gss_OID        display_name_type = GSS_C_NO_OID;

/* Copy the host service principal name
   into the GSS buffer structure */
input_name.length = strlen( szHostName );
input_name.value  = strdup( szHostName );

/* Convert the host service principal name
   into GSS internal form */
gMaj = gss_import_name( &gMin,
                      &input_name,
                      GSS_KRB5_NT_HOSTBASED_SERVICE_NAME,
                      &principal_name );

/* Convert the imported name to human-readable form */
gMaj = gss_display_name( &gMin,
                      principal_name,
                      &display_name,
                      &display_name_type );

*****/

```

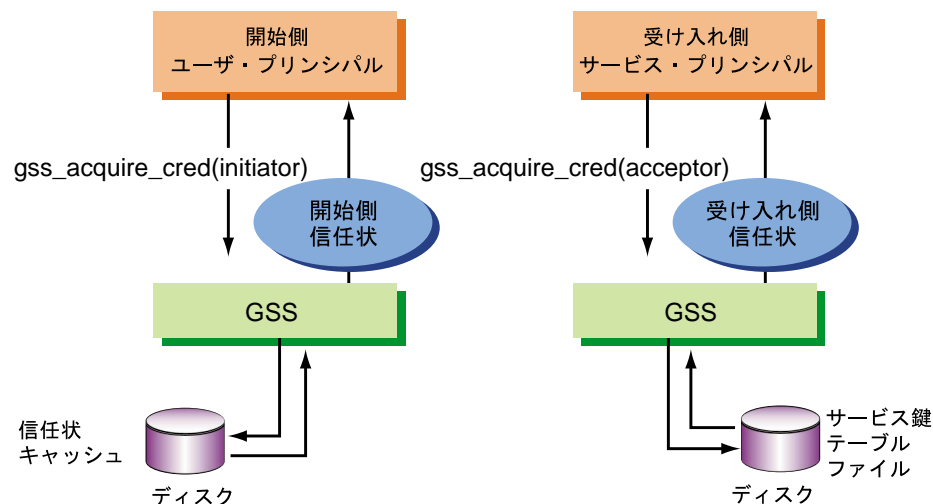
## C.5 手順 2： 信任状の取得

ここでは、開始側と受け入れ側の信任状の取得方法を説明します。信任状には、開始側と受け入れ側が相互に認証するために使う、セキュリティ・サーバからの TGT とサービス鍵テーブル・エントリが含まれています。このため、開始側と受け入れ側の両方が、セキュリティ・コンテキストを確立する前に信任状を取得する必要があります。

要求される信任状のタイプには、次の 3 つがあります。

- GSS\_C\_INITIATE — `gss_init_sec_context()` を呼び出すことによってセキュリティ・コンテキストを開始するアプリケーション用です。
- GSS\_C\_ACCEPT — `gss_accept_sec_context()` を呼び出すことによってセキュリティ・コンテキストを受け入れるアプリケーション用です。
- GSS\_C\_BOTH — セキュリティ・コンテキストの開始と受け入れの両方を行うアプリケーション、つまり、クライアントとサーバの両方の役割を果たすアプリケーション用です。

以下に示すように、信任状の取得はローカルな処理です。ピア・アプリケーションとの通信は必要ありません。



ZK-1828U-AI

`gss_acquire_cred()` 関数は、インポートされた名前を使って、ユーザまたはサービス名に対応する有効なチケットを探します。信任状は、開始側の場合は HP の信任状キャッシュの中に、受け入れ側の場合はサービス鍵テーブルの中に存在していなければなりません。ただし、受け入れ側アプリケーションが、他の受け入れ側アプリケーションとのセキュリティ・コンテキストの開始も行う場合は例外です。たとえば、プロキシがその例です。この場合、信任状キャッシュとサービス鍵テーブル・ファイル・エントリの両方がどちらかの役割をサポートする必要があります。

サービス鍵テーブル・ファイルの省略時の名前は、Windows と UNIX のどちらのプラットフォームでも `v5srvtab` です。

`gss_acquire_cred()` が `GSS_C_INITIATE` 信任状を要求する開始側によって呼び出されると、この関数は渡されたプリンシパル名を調べ、信任状キャッシュの中で対応する TGT を探します。プリンシパル名の指定がない場合、この関数は、信任状キャッシュの省略時のプリンシパル名と同じ省略時の名前を使って、信任状キャッシュから有効な TGT を探します (ユーザのログイン名は使用されません)。

`gss_acquire_cred()` が `GSS_C_ACCEPT` 信任状を要求する受け入れ側によって呼び出されると、この関数は渡されたプリンシパル名を調べ、サービス鍵テーブル・ファイルの中で対応するエントリを探します。プリンシパル名の指定がない場合、この関数は、`host/fqdn@REALM` の省略時の受け入れ側のプリンシパル名に対応する、サービス鍵テーブル・ファイル・エントリを探します。

セキュリティ・コンテキストの開始側にも受け入れ側にもなるアプリケーションは、信任状キャッシュとサービス鍵テーブルの両方を保持している必要があります。この場合、`gss_acquire_cred()` は、`GSS_C_INITIATE` 信任状を要求するアプリケーションによって呼び出されると、プリンシパル名が渡されたことを確認し、このプリンシパル名に対応する有効な TGT が信任状キャッシュにあるかどうか探します。信任状キャッシュに TGT がなければ、この関数はサービス鍵テーブルで該当するエントリを探します。サービス鍵テーブルにエントリがない場合は、テーブル・ファイルの鍵を使って KDC から TGT を取得します。

---

#### 注意

---

開始側からの `gss_acquire_cred()` の呼び出しは省略可能です。`gss_init_sec_context()` が呼び出されるまで開始側が `gss_acquire_cred()` を呼び出さない場合、名前が省略時のものでタイプが `GSS_C_INITIATE` である信任状が取得されます。

受け入れ側からの `gss_acquire_cred()` の呼び出しもまた省略可能です。この場合、`gss_accept_sec_context()` は、名前が省略時のものでタイプが `GSS_C_ACCEPT` である信任状を取得します。受け入れ側アプリケーションが `gss_acquire_cred()`

で信任状を取得し、必要に応じてこれを再利用すると、性能の向上につながります。

`gss_acquire_cred()` では次のパラメータを使用します。

入力パラメータ	
<code>desired_name</code>	信任状を要求しているプリンシパルまたはサービスの名前。指定されなければ、省略時のプリンシパルが使用されます。
<code>time_req</code>	要求された信任状の存続期間(無視されます)。信任状はすでに存在しているので、信任状の存続期間は指定できません。存続期間は、信任状が作成された状況によって決まります。
<code>desired_mechs</code>	要求されているセキュリティ・メカニズム (Kerberos 5)。
<code>cred_usage</code>	信任状の用途 (開始, 受け入れ, または両方)。
出力パラメータ	
<code>minor_status</code>	Kerberos 5 エラー・コード。
<code>output_cred_handle</code>	取得した信任状。
<code>actual_mechs</code>	使用されているセキュリティ・メカニズム (Kerberos 5)。
<code>time_rec</code>	信任状の存続期間が戻されます。これは、開始側でのみサポートされています。受け入れ側の存続期間は不定です。

サンプル・プログラムから抜粋した下記のコードは、開始側用に `GSS_C_INITIATE` 信任状を、受け入れ側用に `GSS_C_ACCEPT` 信任状を取得する方法を示しています。プリンシパル名は手順 1 でインポートされています。

```
/******\n\nOM_uint32      gMaj, gMin          = 0;\nOM_uint32      cred_lifetime_requested = 600;\nOM_uint32      cred_lifetime_received  = 0;\ni_32          cred_usage           = GSS_C_INITIATE;\ngss_cred_id_t  cred_handle          = GSS_C_NO_CREDENTIAL;\ngss_OID_set    actual_mechs          = GSS_C_NO_OID_SET;\n\n/* Get an Initiator credential from the cred cache */\ngMaj = gss_acquire_cred( &gMin,
```

```

        principal_name,
        cred_lifetime_requested,
        rfc_krb5_c_OID_set,
        cred_usage,
        &cred_handle,
        &actual_mechs,
        &cred_lifetime_received );
:
:
:

OM_uint32      gMaj, gMin          = 0;
OM_uint32      cred_lifetime_received = 0;
i_32          cred_usage          = GSS_C_ACCEPT;
gss_cred_id_t  cred_handle         = GSS_C_NO_CREDENTIAL;
gss_OID_set    actual_mechs        = GSS_C_NO_OID_SET;

/* Get an Acceptor credential from the cred cache
   or from the v5srvtab key table file */
gMaj = gss_acquire_cred( &gMin,
                        principal_name,
                        GSS_C_INDEFINITE,
                        GSS_C_NO_OID_SET,
                        cred_usage,
                        &cred_handle,
                        &actual_mechs,
                        &cred_lifetime_received );

/*****\

```

信任状を取得した後は、`gss_release_cred()` 関数を使って、割り当てられていた記憶域を解放する必要があります。サンプル・プログラムにこの例が含まれています。

一般に、バッファがユーザによって (`malloc` や `strdup` などを使って) ローカルに割り当てられた場合は、アプリケーションがこれを解放する必要があります。バッファが GSS-API 関数によって割り当てられた場合は、`gss_release_xxx()` 関数を使って解放する必要があります。これらのアクションはメモリ・リークを防ぎます。

「信任状管理関数」で、他の信任状関数を使ってできる操作について示してあります。

## C.6 手順 3：セキュリティ・コンテキストの確立

アプリケーションとそのピアが信任状を取得した後、セキュリティ・コンテキストを確立できます。セキュリティ・コンテキストとは基本的に、アプリケーションとそのピアとの間の対話を保護する、一連のセキュリティ・パラメータです。

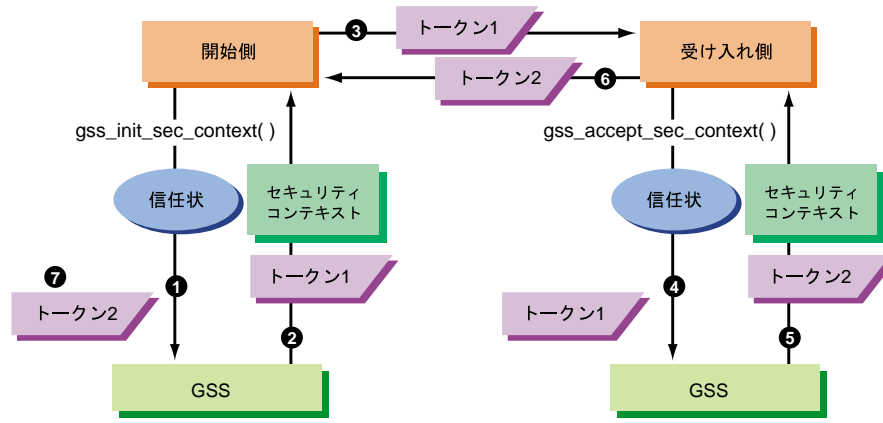
アプリケーションとそのピアの間に、複数のセキュリティ・コンテキストが同時に存在することがあります。ただし、1 つのコンテキストを、複数の対

話に対して使用することはできません。たとえば、クライアント/サーバー・アプリケーションで、あるサーバー・アプリケーションと 10 のクライアントが通信している場合は、各クライアントがそれぞれ一意のセキュリティ・コンテキストを持ち、サーバは 10 のセキュリティ・コンテキストを、各クライアントに対して 1 つずつ管理することになります。

コンテキストの開始時に、アプリケーションは「セキュリティ・コンテキスト管理関数」で説明されている各種のセキュリティ・オプションを選択できます。

- チャンネル・バインディング
- 機密性と完全性
- リプレイ検出
- 順序外メッセージ検出
- 相互認証
- 暗号化アルゴリズム (DES または DES3)
- チケットの転送 (信任状の委任)

アプリケーションがセキュリティ・コンテキストを確立するときの手順は、次のとおりです。開始側と受け入れ側によって使われる関数呼び出しのいくつかは、コンテキストを完全に確立するために必要になります。



ZK-1829U-AI

1. 開始側は `gss_init_sec_context()` を呼び出すことによって受け入れ側とのセキュリティ・コンテキストを開始し、以前に取得した信任状と、受け入れ側によって使われる識別情報を渡します。
2. `gss_init_sec_context()` 関数は、部分的に作成されたセキュリティ・コンテキストとトークン、およびステータス・コードを戻します。  
このトークンには、受け入れ側に送信する必要のある GSS-API 情報が入っています。トークンは開始側アプリケーションから見ると不透過ですが、アプリケーションはトークンの長さを認識しているため、受け入れ側にトークンを送信できます。  
  
関数を再度呼び出す必要があるかどうかはステータス・コードによって示されます。関数を再度呼び出す場合は、受け入れ側からリターン・トークンを受け取り、関数に渡さなくてはなりません。  
  
エラー・ステータスが戻されるときに、トークンが戻されることがあります。戻されたステータス・コードに関係なく、トークンは相手側のアプリケーションに送信される必要があります。
3. 開始側は通信プロトコルを通じて、コンテキストの確立を待っている受け入れ側にトークンを送信します。  
  
トークンがどのようにして送信されるかは、アプリケーションに依存します。ただし GSS-API 標準では、コンテキスト確立時にトークンが生成された順番でトークンを送信するように定められています。



4. 受け入れ側は、通信プロトコルからトークンを読み取り、  
`gss_accept_sec_context()` を呼び出して、以前に取得した信任状と、受け取ったトークンを渡します。
5. `gss_accept_sec_context()` 関数は、ステータス・コードを戻し、場合によってはトークンも戻します。  
  
関数を再度呼び出す必要があるかどうかはステータス・コードによって示されます。関数を再度呼び出す場合は、開始側からリターン・トークンを受け取り、関数に渡さなくてはなりません。  
  
エラー・ステータスが戻されるときに、トークンが戻されることがあります。戻されたステータス・コードに関係なく、トークンは相手側のアプリケーションに送信される必要があります。
6. `gss_accept_sec_context()` がトークンを戻した場合、受け入れ側は、セキュリティ・コンテキストの確立を続行するためにこのトークンを待っている開始側に、通信プロトコルを通じてトークンを送信する必要があります。
7. 開始側はリターン・トークンを受け取ると、`gss_init_sec_context()` を呼び出してトークンを渡します。

手順 1 から 5 までは、開始側と受け入れ側の最低限のやり取りを示しています。手順 6 と 7 は、`gss_init_sec_context()` および `gss_accept_sec_context()` 関数の複数の呼び出しを例示しています。これらの関数は、メジャー・ステータス・コード `GSS_S_CONTINUE` が戻されると、ループに入ります。`gss_init_sec_context()` 関数からトークンが戻されたら、受け入れ側に戻す必要があります。`GSS_S_CONTINUE` が戻されると、受け入れ側からトークンが戻ってくるものと期待されるため、次の `gss_init_sec_context()` の呼び出しへの入力として渡されなければなりません。この動作は通常、Kerberos が相互認証を実行するときに発生します。

`GSS_S_CONTINUE` ステータスがメジャー・ステータス・コードとして戻され、呼び出しによって戻されたトークンがピア・アプリケーションに送信されると、ローカル・アプリケーション側でセキュリティ・コンテキストが完全に確立します。

`gss_init_sec_context()` 関数は次のパラメータを使います。

入力パラメータ	
<code>initiator_cred_handle</code>	開始側の信任状 (または信任状キャッシュからの省略時の信任状)。
<code>target_name</code>	受け入れ側の名前。
<code>mech_type</code>	要求されているセキュリティ・メカニズム (Kerberos 5)。
<code>req_flags</code>	サービス・オプション。たとえば、DES3/DES 暗号化、信任状の委任、相互認証、リプレイ・メッセージの検出、順序外メッセージの検出など。
<code>time_req</code>	要求されたコンテキストの存続期間 (無視されます)。
<code>input_chan_bindings</code>	チャネル・バインディング
<code>input_token</code>	受け入れ側から戻されたトークン。
出力パラメータ	
<code>minor_status</code>	Kerberos 5 エラー・コード。
<code>context_handle</code>	開始するセキュリティ・コンテキスト。
<code>actual_mech_type</code>	使用されているセキュリティ・メカニズム (Kerberos 5)。
<code>output_token</code>	受け入れ側に送信されるトークン。
<code>ret_flags</code>	このコンテキストがサポートしているサービス・オプションを識別するフラグ。
<code>time_rec</code>	受け取ったコンテキストの存続期間 (常に不定です)。

サンプル・プログラムから抜粋した下記のコードは、`gss_init_sec_context()` を呼び出して、次のオプションを使ってセキュリティ・コンテキストを確立する方法を示しています。

- 相互認証
- リプレイ検出
- 順序外メッセージ検出

暗号化方式は指定されていないため、省略時の設定で、プリンシパル・データベースの信任状に割り当てられている暗号化方式に設定されます。

```

/*****/

char          szHostName[]      = "host@dev007.cybersafe.com";
OM_uint32     gMaj, gMin        = 0;
OM_uint32     req_flags         = GSS_C_MUTUAL_FLAG |
                                GSS_C_REPLAY_FLAG |
                                GSS_C_SEQUENCE_FLAG;

OM_uint32     req_time          = 3600;
OM_uint32     ret_flags         = 0;
OM_uint32     ret_time;         = 0;
gss_buffer_desc input_name      = GSS_C_EMPTY_BUFFER;
gss_buffer_desc input_token     = GSS_C_EMPTY_BUFFER;
gss_buffer_desc output_token   = GSS_C_EMPTY_BUFFER;
gss_buffer_t   token_handle     = GSS_C_NO_BUFFER;
gss_ctx_id_t   context_handle   = GSS_C_NO_CONTEXT;
gss_name_t     service_name     = GSS_C_NO_NAME;
gss_OID        actual_mech_type = GSS_C_NO_OID;

/* Copy the host service principal name
   into the GSS buffer structure */
input_name.length = strlen( szHostName );
input_name.value  = strdup( szHostName );

/* Convert the host service principal name
   into GSS internal form */
gMaj = gss_import_name( &gMin,
                        &input_name,
                        GSS_KRB5_NT_HOSTBASED_SERVICE_NAME,
                        &service_name );

/* Establish a context. The default claimant credential is
   specified. In that case, the credential is pulled from the
   credential cache. Alternatively, the claimant credential
   could have been acquired via gss_acquire_cred and passed
   to gss_init_sec_context. */
do {
    token_handle = input_token.length ?
        &input_token : GSS_C_NO_BUFFER;

    /* Initiate a security context with the Acceptor */
    gMaj = gss_init_sec_context( &gMin,
                                GSS_C_NO_CREDENTIAL,
                                &context_handle,
                                service_name,
                                GSS_C_NO_OID,
                                req_flags,
                                req_time,
                                GSS_C_NO_CHANNEL_BINDINGS,
                                token_handle,
                                &actual_mech_type,
                                &output_token,
                                &ret_flags,
                                &ret_time);

    /* If there is a token to send, regardless of
       error condition, then send it to the peer */
    if ( output_token.length != 0 ) {
        SendToken( &output_token );
    }

    /* Clear the buffers for the next iteration */
    Local_Release_Buffer( &input_token );
    gss_release_buffer( &tmpMinor, &output_token );
} while ( gMaj != GSS_S_COMPLETE );

```

```

/* Now is the proper time to check the status code */
if ( GSS_ERROR( gMaj ) {
    /* Process error and exit */
}

/* If context establishment requires a token in response,
   then fetch it from the peer */
if ( gMaj & GSS_S_CONTINUE_NEEDED ) {
    RecvToken( &input_token );
}

} while ( gMaj & GSS_S_CONTINUE_NEEDED );

/*****

```

コンテキストが不要になったら、`gss_delete_sec_context()` 関数を使って破壊する必要があります。サンプル・プログラムにこの例が含まれています。

`gss_accept_sec_context()` 関数は次のパラメータを使います。

入力パラメータ	
<code>context_handle</code>	確立しているセキュリティ・コンテキスト
<code>acceptor_cred_handle</code>	受け入れ側への信任状。
<code>input_token_buffer</code>	開始側からのトークン。
<code>input_chan_bindings</code>	チャンネル・バインディング。
出力パラメータ	
<code>minor_status</code>	Kerberos 5 エラー・コード。
<code>src_name</code>	開始側の名前。
<code>mech_type</code>	使用されているセキュリティ・メカニズム (Kerberos 5)。
<code>output_token</code>	開始側に送信されるトークン。
<code>ret_flags</code>	このコンテキストがサポートしているサービス・オプションを識別するフラグ。
<code>time_rec</code>	受け取ったコンテキストの存続期間 (常に不定です)。
<code>delegated_cred_handle</code>	コンテキストの開始側から委任された信任状。

サンプル・プログラムから抜粋した下記のコードは、`gss_accept_sec_context()` を呼び出して、セキュリティ・コンテキストを確立する方法を示しています。

```

*****

OM_uint32      gMaj, gMin      = 0;
OM_uint32      ret_flags      = 0;
OM_uint32      ret_time       = 0;
OM_uint32      ctx_flags      = 0;
gss_buffer_desc input_token    = GSS_C_EMPTY_BUFFER;
gss_buffer_desc output_token   = GSS_C_EMPTY_BUFFER;
gss_cred_id_t   delegated_cred = GSS_C_NO_CREDENTIAL;
gss_ctx_id_t    context_handle = GSS_C_NO_CONTEXT;
gss_name_t      src_name       = GSS_C_NO_NAME;
gss_OID         actual_mech_type = GSS_C_NO_OID;

/* Loop based on the example in
   "draft-ietf-cat-gssv2-cbind-07.txt" */
do {
    OM_uint32 tmpMinor = 0;

    /* Fetch next security context token from the Initiator */
    RecvToken( &input_token );

    /* Wait for the Initiator to try to establish a security
       context. The acceptor will use a previously acquired
       acceptor credential. */
    gMaj = gss_accept_sec_context( &gMin,
                                   &context_handle,
                                   cred_handle,
                                   &input_token,
                                   GSS_C_NO_CHANNEL_BINDINGS,
                                   &src_name,
                                   &actual_mech_type,
                                   &output_token,
                                   &ret_flags,
                                   &ret_time,
                                   &delegated_cred );

    /* If the authentication routine forms a token to send to
       the Initiator, then send it now -- before function error
       checking, so that if an error token is generated, the
       Initiator will receive it. */
    if ( output_token.length != 0 ) {
        SendToken( &output_token );
    }

    /* Clear the buffers for the next iteration */
    Local_Release_Buffer( &input_token );
    gss_release_buffer( &tmpMinor, &output_token );

    /* Now is the proper time to check the status code */
    if ( GSS_ERROR( gMaj ) ) {
        /* Process error and exit */
    }

} while ( gMaj & GSS_S_CONTINUE_NEEDED );

*****

```

コンテキストが不要になったら、`gss_delete_sec_context()` 関数を使って破壊する必要があります。サンプル・プログラムにこの例が含まれています。

## C.7 手順 4：メッセージの交換

アプリケーションとそのピアの間で信任状とセキュリティ・コンテキストが取得された後で、メッセージを保護し、送信できます。ここでは、Application Security SDK で使用可能な、メッセージの保護に関するオプションについて説明します。

GSS-API 標準の Version 2 には、データを保護する次の 2 組の関数があります。

- `gss_get_mic()` と `gss_verify_mic()` は、メッセージの完全性を提供します。
- `gss_wrap()` と `gss_unwrap()` は、メッセージの完全性と、オプションで機密性も提供します。

送信側のアプリケーションがこれらの関数の 1 つを呼び出すと、その関数によってトークンが戻されます。戻されたトークンには、完全性アルゴリズムを使って送信されているメッセージに対して計算された署名が含まれています。`gss_wrap()` の場合、トークンにはメッセージ自体も含まれます。その場合はオプションで、機密性のためにメッセージを暗号化できます。

Application Security SDK は、次の 4 つの完全性アルゴリズムを提供しています。

- DES3 MD5
- DES MAC
- DES MD5
- MD5

暗号化アルゴリズムは、DES3 と DES の 2 つが提供されています。

アプリケーションはその後、TCP/IP などの通信プロトコルを使って、受信側のアプリケーションにトークン（および、このトークンにメッセージが含まれていなければメッセージ）を送ります。受信側のアプリケーションは、この関数の対となっている関数を呼び出して、渡されたデータに対して同様の操作を実行します。

これらの関数は、データを保護するために、次の3つのオプションを提供しています。

1. メッセージと署名を別々に送信する。

このオプションを使うと、トークンには、送信側のアプリケーションによって `gss_get_mic()` 関数を使って作成された署名が含まれます。メッセージはトークンには含まれません。

受信側のアプリケーションは、トークンとメッセージを受け取ると、`gss_verify_mic()` 関数を呼び出して署名をチェックし、メッセージの送信中にメッセージと署名が書き換えられていないことを確認します。

2. メッセージと署名を一緒に送信する。

このオプションを使うと、トークンには、送信側のアプリケーションによって `gss_wrap()` 関数を使って作成された署名が含まれます。メッセージはトークンの中にカプセル化されます。

受信側のアプリケーションは、トークンとメッセージを受け取ると、`gss_unwrap()` 関数を呼び出して署名をチェックし、メッセージの送信中にメッセージと署名が書き換えられていないことを確認します。

3. 暗号化されたメッセージと署名を一緒に送信する。

これは、オプション2と同じですが、メッセージと署名は `gss_wrap()` 関数によって暗号化され、`gss_unwrap()` 関数によって復号されます。

各オプションは、分散型アプリケーションのデータを保護するときの特定の用途に対応するものです。

たとえば、アプリケーションに、保護する必要のあるデータと、保護する必要のないデータが含まれている場合、アプリケーションは保護すべきデータのみを保護するように選択できます。これらの判断基準は、データの長さや速度、効率、暗号化アルゴリズムの強度に応じて、アプリケーションに依存します。データ量が多い場合は、メッセージの一部だけを保護するのが効率的です。ただしメッセージ全体を保護することはめったにありません。

アプリケーションが `gss_wrap()` と機密性を使っている例を以下に示します。

送信先アプリケーションに送る情報：

氏名,  
ソーシャル・セキュリティ番号,  
クレジット・カード番号,  
住所,  
電話番号,  
etc.

メッセージは次のようになります：

Bob Smith, XXXXXXXXXXXX, 住所, 電話番号, etc.

↑  
SSN とカード番号のみ暗号化

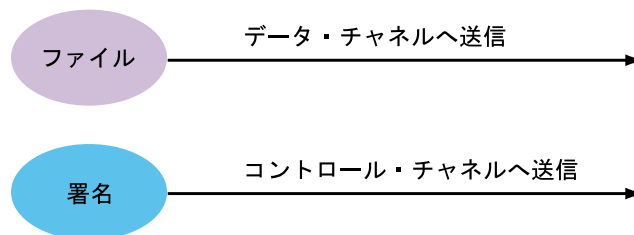
ZK-1830U-AI

アプリケーションがメッセージの送信中にその完全性が保護されたことを保証し、さらに、データのあるチャンネルで送信し、チェックサムを ftp など別のアウト・オブ・バウンド (制御) チャンネルで送信する必要がある場合、アプリケーションは下に示す `gss_get_mic()` 関数を使う必要があります。

送信先アプリケーションに送る情報：

電子メール・メッセージのテキスト

メッセージは次のようになります：



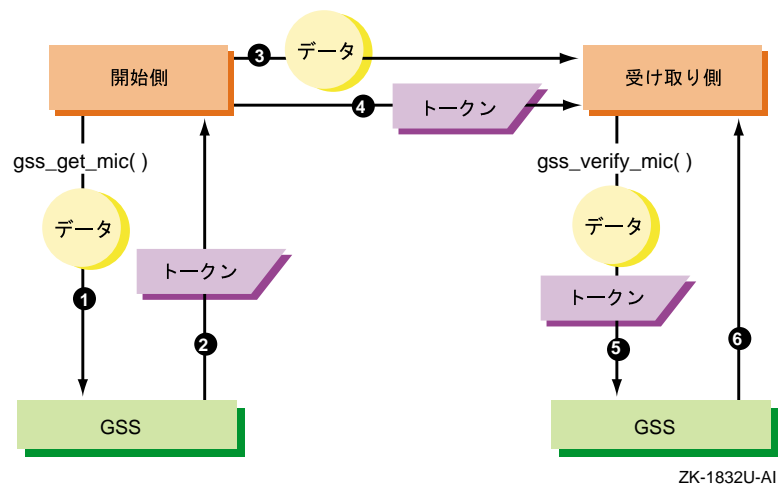
ZK-1831U-AI



### C.7.1 gss\_get\_mic( ) と gss\_verify\_mic( ) の使用

gss\_get\_mic( ) を呼び出すと、アプリケーションは、ピア・アプリケーションが gss\_verify\_mic( ) を使ってチェックできる、データの署名を作成できます。これにより、メッセージの完全性、つまりメッセージとその署名が送信中に書き換えられていないことを保証できます。

これらの関数を使って開始側から受け入れ側へ、メッセージを送信する一連の手順は次のとおりです。



1. 開始側が gss\_get\_mic( ) を呼び出し、署名を生成する対象のデータを渡します。
2. この関数は、渡したデータの署名の入ったトークンを戻します。ただしデータ自体はトークンには含まれていません。
3. 開始側が受け入れ側にデータを送信します。  
トークンとデータは、任意の順番での送信、別々のメッセージとしての送信、同じメッセージ内での送信、またはその他のオプションを指定しての送信が可能です。
4. 開始側が受け入れ側へトークンを送ります。
5. 受け入れ側は gss\_verify\_mic( ) を呼び出し、開始側から受け取ったデータとトークンを渡します。
6. 受け入れ側の GSS が署名を検査し、メッセージの完全性を確認します。

6. `gss_verify_mic()` 関数は、署名とデータを突合せたチェックが正常に終了したかどうかを示すステータスを戻します。

`gss_get_mic()` 関数には次のパラメータがあります。

入力パラメータ	
<code>context_handle</code>	使用しているセキュリティ・コンテキスト。
<code>qop_req</code>	要求された Quality Of Protection (QOP)。
<code>message_buffer</code>	保護するメッセージ。
出力パラメータ	
<code>minor_status</code>	Kerberos 5 エラー・コード。
<code>message_token</code>	署名の入ったトークン。

サンプル・プログラムから抜粋した下記のコードは、この関数呼び出しを使ってメッセージを送信する方法を示しています。省略時の QOP が指定されます。この例では、セキュリティ・コンテキストに指定された暗号化タイプによって決まります。

```
/******  
char          szMessage[]   = "This is a test message";  
OM_uint32     gMaj, gMin    = 0;  
gss_buffer_desc input_buffer = GSS_C_EMPTY_BUFFER;  
gss_buffer_desc output_buffer = GSS_C_EMPTY_BUFFER;  
gss_qop_t      qop_req      = GSS_C_QOP_DEFAULT;  
  
/* Stuff plaintext message into the input buffer*/  
input_buffer.value = strdup( szMessage );  
input_buffer.length = strlen( szMessage );  
  
/* Create a message integrity code (MIC) from the message */  
gMaj = gss_get_mic( &gMin,  
                   context_handle,  
                   qop_req,  
                   &input_buffer,  
                   &output_buffer );  
  
/* Send the MIC token */  
SendToken( &output_buffer );  
  
*****
```

トークンに関連付けられている記憶域は、使用後はメモリ・リークを避けるために `gss_release_buffer()` 関数を使って解放する必要があります。

`gss_verify_mic()` 関数には次のパラメータがあります。

---

#### 入力パラメータ

---

context_handle	使用しているセキュリティ・コンテキスト。
message_buffer	チェックするメッセージ。
token_buffer	署名の入ったトークン。

---

#### 出力パラメータ

---

minor_status	Kerberos 5 エラー・コード。
qop_state	使用された QOP。

---

`gss_verify_mic()` 関数から `GSS_S_COMPLETE` が戻されると、署名は正常に検査されたことを示します。

サンプル・プログラムから抜粋した下記のコードは、この関数呼び出しを使ってメッセージを受信する方法を示しています。

```
/******/  
char          szMessage[]    = "This is a test message";  
OM_uint32     gMaj, gMin     = 0;  
gss_buffer_desc input_buffer  = GSS_C_EMPTY_BUFFER;  
gss_buffer_desc message_buffer = GSS_C_EMPTY_BUFFER;  
gss_qop_t     qop_state      = 0;  
  
/* Stuff plaintext message into the message buffer */  
message_buffer.value = strdup( szMessage );  
message_buffer.length = strlen( szMessage );  
  
/* Receive the MIC token */  
RecvToken( &input_buffer );  
  
/* Verify the message against the message integrity code (MIC) */  
gMaj = gss_verify_mic( &gMin,  
                      context_handle,  
                      &message_buffer,  
                      &input_buffer,  
                      &qop_state );  
  
/******/>
```

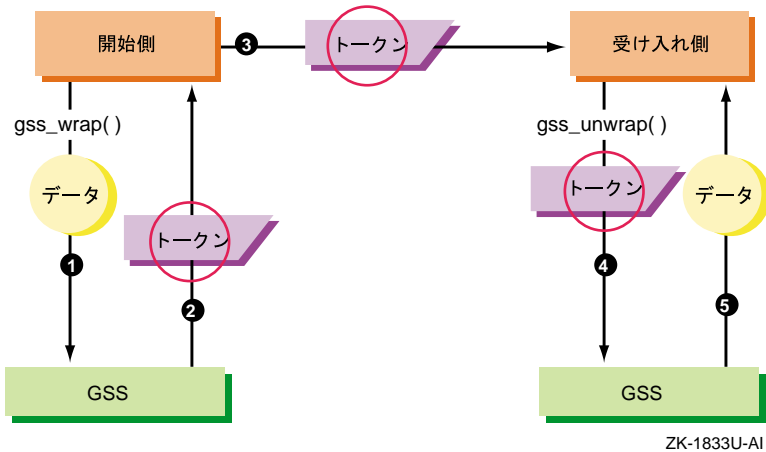
### C.7.2 `gss_wrap()` と `gss_unwrap()` の使用

`gss_wrap()` を呼び出すと、アプリケーションは、ピア・アプリケーションが `gss_unwrap()` を使ってチェックできる、データをラップできます。これにより、メッセージが送信中に書き換えられていないことを保証できます (メッセージの完全性)。

さらに、アプリケーションは、機密性のためにデータを暗号化し、データを誰にも見えないように保証できます。データをトークンにカプセル化する

ると、ネットワーク上で送信されるデータのための安全なエンベロープが作成されます。

これらの関数を使って開始側から受け入れ側へ、メッセージを送信する一連の手順は次のとおりです。



1. 開始側は、トークンにラップしたいデータを使って、`gss_wrap()` を呼び出します。機密性が同時に呼び出されることもあります。
2. データと、そのデータの署名の入ったトークンが戻されます。機密性が要求された場合は、両方とも暗号化されます。
3. 開始側は受け入れ側にトークンを送ります。
4. 受け入れ側は `gss_unwrap()` を呼び出し、開始側から受け取ったデータとトークンを渡します。
5. `gss_unwrap()` 関数は、署名をチェックするステータス・コードと、元のデータの両方を戻します。最初に機密性が要求されていた場合は、データの復号も行われます。

`gss_wrap()` 関数には次のパラメータがあります。

入力パラメータ	
<code>context_handle</code>	使用しているセキュリティ・コンテキスト。
<code>conf_req_flag</code>	機密性の要求 (真または偽)。

---

#### 入力パラメータ

---

qop_req	要求された Quality Of Protection (QOP)。
input_message_buffer	保護するメッセージ。

---

#### 出力パラメータ

---

minor_status	Kerberos 5 エラー・コード。
conf_state	機密性の確認 (真または偽)。
conf_state	保護されたメッセージ。

---

サンプル・プログラムから抜粋した下記のコードは、この関数呼び出しを使って暗号化されたメッセージを送信する方法を示しています。省略時の QOP が指定されます。この例では、セキュリティ・コンテキストに指定された暗号化タイプによって決まります。

```
/******  
char          szMessage[]   = "This is a test message";  
OM_uint32     gMaj, gMin    = 0;  
int           conf_req_flag = 1;  
int           conf_state    = 0;  
gss_buffer_desc input_buffer = GSS_C_EMPTY_BUFFER;  
gss_buffer_desc output_buffer = GSS_C_EMPTY_BUFFER;  
gss_qop_t      qop_req      = GSS_C_QOP_DEFAULT;  
  
/* Stuff plaintext message into the input buffer*/  
input_buffer.value = strdup( szMessage );  
input_buffer.length = strlen( szMessage );  
  
/* Wrap (Seal) the message into a token */  
gMaj = gss_wrap( &gMin,  
                context_handle,  
                conf_req_flag,  
                qop_req,  
                &input_buffer,  
                &conf_state,  
                &output_buffer );  
  
/* Send the message token */  
SendToken( &output_buffer );  
/******
```

保護されたメッセージに関連付けられている記憶域は、使用後はメモリ・リークを避けるために `gss_release_buffer()` 関数を使って解放する必要があります。

`gss_unwrap()` 関数には次のパラメータがあります。

---

#### 入力パラメータ

---

context_handle	使用しているセキュリティ・コンテキスト。
input_message_buffer	ラップ解除, チェック, 必要であれば復号を行うメッセージ。

---

#### 出力パラメータ

---

minor_status	Kerberos 5 エラー・コード。
output_message_buffer	ラップ解除されたメッセージ。
conf_state	機密性の確認 (真または偽)。
qop_state	使用された QOP。

---

gss\_unwrap() 関数から GSS\_S\_COMPLETE が戻されると, 署名は正常に検査されたことを示します。

サンプル・プログラムから抜粋した下記のコードは, この関数呼び出しを使ってメッセージを受信する方法を示しています。

```
/******/  
  
char          szMessage[256];  
OM_uint32     gMaj, gMin    = 0;  
i_32         conf_state    = 0;  
gss_buffer_desc input_buffer = GSS_C_EMPTY_BUFFER;  
gss_buffer_desc output_buffer = GSS_C_EMPTY_BUFFER;  
gss_qop_t     qop_state     = 0;  
  
/* Receive the message token */  
RecvToken( &input_buffer );  
  
/* Unwrap (Unseal) the message token */  
gMaj = gss_unwrap( &gMin,  
                  context_handle,  
                  &input_buffer,  
                  &output_buffer,  
                  &conf_state,  
                  &qop_state );  
  
/* Extract plaintext message from the output buffer*/  
if ( output_buffer.value != NULL ) {  
    strcpy( szMessage, output_buffer.value );  
}  
  
/******/>
```

ラップ解除したメッセージに関連付けられている記憶域は, 使用後はメモリ・リークを避けるために gss\_release\_buffer() 関数を使って解放する必要があります。

# C.8 手順 5：セキュリティ・コンテキストの終了

一方のアプリケーションが、そのピアとの通信を完了すると、セキュリティ・コンテキストを終了することができます。GSS-API 標準では、開始側と受け入れ側のアプリケーションによってセキュリティ・コンテキストを終了できるようになっています。

セキュリティ・コンテキストを終了する手順は、開始側も受け入れ側も同じです。アプリケーションとピアの間に複数のセキュリティ・コンテキストが存在している場合は、個々のコンテキストを別々に終了する必要があります。

`gss_delete_sec_context()` 関数には次のパラメータがあります。

入力パラメータ	
<code>context_handle</code>	削除するセキュリティ・コンテキスト。
出力パラメータ	
<code>minor_status</code>	Kerberos 5 エラー・コード。
<code>output_token</code>	ローカルでの削除を要求する GSS_C_NO_BUFFER を指定します。 HP ではこれを推奨しています。

サンプル・プログラムから抜粋した下記のコードは、セキュリティ・コンテキストを終了し、以前に取得した信任状を解放する方法を示しています。

```

/*****
/* The V2 GSS_API specs recommend setting the output token
   parameter to NULL to signify that no token is to be
   returned. */

gMaj = gss_delete_sec_context( &gMin,
                               &context_handle,
                               GSS_C_NO_BUFFER );

context_handle = GSS_C_NO_CONTEXT;
:
:
:
..gMaj = gss_release_cred( &gMin,
                           &cred_handle );

cred_handle = GSS_C_NO_CREDENTIAL;

*****/
```

## C.9 高度な概念

GSS-API 標準に対する HP のエクステンションにより、アプリケーション側でそのセキュリティ環境をより管理できるようになります。これらのエクステンションでは、次のことが可能です。

- ユーザ用に初回信任状を取得する
- 開始側、受け入れ側、セキュリティ・サーバの間で必要な時間の同期を設定する
- 機密性の向上のために DES3 暗号化方式を使う

これらのエクステンションは、メカニズムと実装に固有です。エクステンションを使うと、アプリケーションの移植性に影響を及ぼすことがあります。たとえば、HP の DES3 の実装は、他の GSS-API ベンダの DES3 との相互運用性がありません。

### C.9.1 初回信任状の取得

標準の GSS-API 関数を使って Kerberos 5 を扱うときには、アプリケーションは GSS-API によって保護したアプリケーションを実行する前に、チケット・グランティング・チケット (TGT) を取得している必要があります。ユーザは、`kinit` またはこれに相当するアプリケーション (たとえば、HP の `Single SignOn`) を実行することによって TGT を取得できます。

いくつかの状況では、ユーザに、TGT の取得機能を備えたプログラムを 1 つだけ実行させるのが望ましいこともあります。このため、`Application Security SDK` には、セキュリティ・サーバから初回信任状を取得するために使用できるエクステンション関数が含まれています。

`csf_gss_acq_user()` 関数は、秘密鍵または公開鍵認証と、オプションで DES3 暗号化を使ってアプリケーション内から初回信任状を取得する HP のエクステンションです。

この関数は、ユーザ・コンテキストを確立するときに、このユーザの信任状キャッシュがなければこれを作成します。信任状キャッシュが存在していても、要求された信任状に一致するものがない場合、キャッシュは再初期化されます。この場合、アプリケーションは他のアプリケーションがこの信任状キャッシュを使っていないことを確認する必要があります。



この関数は、アプリケーションが表示しなければならないプロンプトとラベルも提供します。アプリケーションは、結果の応答を収集して、それを GSS に戻す必要があります。

`csf_gss_acq_user()` 関数には次のパラメータがあります。

入力パラメータ	
<code>user_name</code>	初回信任状を必要とするユーザまたはサービスの名前。指定がなければ、省略時のプリンシパルまたはログイン名を使用できます。
<code>desired_mechs</code>	要求されているセキュリティ・メカニズム (Kerberos 5)。
<code>options</code>	初回信任状のプロトコル・キー、属性 (存続期間、延長可能、転送可能、事前認証、代理可能)、および暗号化方法 (DES または DES3) を決定するオプション。
<code>user_response</code>	ユーザからの応答をアプリケーションのプロンプトに渡します。
出力パラメータ	
<code>minor_status</code>	Kerberos 5 エラー・コード。
<code>user</code>	取得されているユーザ・コンテキスト。
<code>user_prompt</code>	プロンプトをアプリケーションに渡します。
<code>user_label</code>	プロンプト情報をアプリケーションに渡します。
<code>prompt_state</code>	プロンプト表示のヒントをアプリケーションに渡します。
<code>prompting_mech</code>	使用されているセキュリティ・メカニズム (Kerberos 5)。
<code>pwd_exp_time</code>	ユーザのパスワードの有効期限。

`csf_gss_acq_user()` 関数は、次のように使います。

- セキュリティ・メカニズムとして Kerberos 5 を指定します。
- TGT に対して使いたいオプションを選択します。
- プロンプトに応答します。この関数は、プロンプトをどのように処理するかと、ユーザに求める入力を識別するパラメータを戻します。

この関数はループ関数です。つまり、複数回呼び出す必要が生じることもあります。

- 関数から GSS\_S\_COMPLETE が戻された場合、関数はタスクを完了しています。
- 関数から GSS\_S\_CONTINUE\_NEEDED が戻された場合、Kerberos 5 からの 1 つまたは複数のプロンプトを満足する必要があります。

他にも、初回信任状の管理に利用できる特殊な HP 関数があります。たとえば、csf\_gss\_inq\_user() は TGT の問い合わせを行います。不要になったユーザ・コンテキストに対応する記憶域を解放するには、csf\_gss\_release\_user() を使います。

サンプル・プログラムから抜粋した下記のコードは、次のオプションを使って初回信任状のためのユーザ・コンテキストを確立する方法を示しています。

- 転送可能
- 10 時間の存続期間
- 48 時間の延長期限
- DES3 暗号化

また、パスワード、チャレンジ、ワン・タイム・パスワード (OTP) のプロンプトがあるかどうかもチェックします。プリンシパル名は以前にインポートされています。不要になった記憶域を解放するための呼び出しは示していません。

```
/******  
OM_uint32      gMaj      = 0;  
OM_uint32      gMin      = 0;  
i_32          done      = 0;  
i_32          prompt_state = CSF_GSS_C_USER_STATE_NULL;  
int           optCount   = 0;  
gss_buffer_desc user_prompt1 = GSS_C_EMPTY_BUFFER;  
gss_buffer_desc user_label1  = GSS_C_EMPTY_BUFFER;  
gss_buffer_desc response1    = GSS_C_EMPTY_BUFFER;  
gss_buffer_t    user_prompt  = &user_prompt1;  
gss_buffer_t    user_label    = &user_label1;  
gss_buffer_t    response      = GSS_C_NO_BUFFER;  
csf_gss_user_t   gss_user     = CSF_GSS_C_NO_USER;  
gss_OID         prompting_mech = GSS_C_NO_OID;  
csf_gss_mech_opt_desc opts[10];  
char            buff[256];  
time_t          pwd_exp_time;  
  
/* Set up the csf_gss_acq_user options */  
opts[optCount].mechOID = rfc_krb5_c_OID;  
opts[optCount].id      = CSF_GSS_C_ACQ_USER_OPT_FORWARDABLE;  
++optCount;
```

```

opts[optCount].mechOID = rfc_krb5_c_OID;
opts[optCount].id = CSF_GSS_C_ACQ_USER_OPT_LIFETIME;
opts[optCount].val = "10h";
++optCount;

opts[optCount].mechOID = rfc_krb5_c_OID;
opts[optCount].id = CSF_GSS_C_ACQ_USER_OPT_RENEWABLE;
opts[optCount].val = "48h";
++optCount;

opts[optCount].mechOID = rfc_krb5_c_OID;
opts[optCount].id = CSF_GSS_C_ACQ_USER_OPT_DES3;
++optCount;

opts[optCount].mechOID = GSS_C_NO_OID;

/* Acquire the user context */
do {
    gMaj = csf_gss_acq_user( &gMin,
                           principal_name,
                           rfc_krb5_c_OID_set,
                           opts,
                           response,
                           &gss_user,
                           user_prompt,
                           user_label,
                           &prompt_state,
                           &prompting_mech,
                           &pwd_exp_time );

    /* Check whether user needs to provide more information */
    if ( gMaj & GSS_S_CONTINUE_NEEDED ) {

        switch ( prompt_state )
        {
            case CSF_GSS_C_USER_STATE_PASSWORD_NOECHO:
            case CSF_GSS_C_USER_STATE_PASSWORD_ECHO:
                printf( "Password information is requested...\n");
                break;

            case CSF_GSS_C_USER_STATE_CHALLENGE_NOECHO:
            case CSF_GSS_C_USER_STATE_CHALLENGE_ECHO:
                printf( "Challenge information is requested...\n");
                break;

            case CSF_GSS_C_USER_STATE_OTP_NOECHO:
            case CSF_GSS_C_USER_STATE_OTP_ECHO:
                printf( "OTP information is requested...\n");
                break;

            default:
                printf( "Unrecognized prompt state.\n");
        }

        printf( "%s\n", user_label->value);
        printf( "Enter %s", user_prompt->value);
        fflush(stdout);

        response = &response1;

        fgets(buff, 256, stdin);
        response->value = buff;
    }
} while ( gMaj & GSS_S_CONTINUE_NEEDED );

```

```

        response->length = strlen(buff);
        buff[response->length - 1] = '\0';
    }
    else {
        done = 1;
    }
} while (!done);

/*****

```

## C.9.2 必要な時刻同期

受け入れ側アプリケーションのホストは、信任状を付与するセキュリティ・サーバとの時間のずれが 5 分以内である必要があります。受け入れ側アプリケーションのホストとセキュリティ・サーバとの時間のずれが 5 分以内ならば、受け入れ側アプリケーションとのずれも 5 分以内である必要があります。クロック・スキューと呼ばれるずれが 5 分を超えている場合、認証は失敗します。あるアプリケーションからそのピアへ信任状を転送しているときは、すべてのシステムのクロックを 5 分以内で同期させる必要があります。

ホストのクロックで 5 分の差を許容しているため、チケットの存続期間の残りが時差の上限に満たない場合、セキュリティ・サーバは実際の有効期限よりも前にチケットを拒否することがあります。時差の制限に応じて必要となる実際の有効存続期間は、開始側のクロックがセキュリティ・サーバのクロックよりも進んでいるのか、遅れているのかによって異なりますが、範囲は 1 分から 5 分です。時差の上限に関する GSS の失敗の例として、関数 `gss_init_sec_context()` は、開始側の Kerberos 信任状の有効期限が時差の範囲内にあると失敗し、`context_handle` パラメータを `GSS_C_NO_CONTEXT` に設定します。これが発生すると、この関数は Kerberos エラー、“Credentials Not Found (信任状が見つかりません)” (16 進値 C0001507) を戻します。この場合、開始側アプリケーションは、`gss_init_sec_context()` を使って受け入れ側アプリケーションとのセキュリティ・コンテキストを確立する前に、新しい初回信任状 (TGT) をセキュリティ・サーバから取得する必要があります。

## C.9.3 DES3 暗号化の使用

Application Security SDK は、DES および DES 3 暗号化の両方をサポートしています。ただし、単一のセキュリティ・コンテキストに対して複数の暗号化システムは許可されていません。

DES3 暗号化を使ってメッセージを暗号化できるようにするには、次の条件が満たされている必要があります。

- データベースを作成するときに、TrustBroker Security Server を DES3 向けに設定されていること。
- 開始側と受け入れ側のアプリケーションのプリンシパルが、プリンシパル・データベースで DES3 に対応していること。つまり、ユーザが開始側アプリケーションを実行している場合は、ユーザのプリンシパルが DES3 に対応していなければなりません。
- 開始側アプリケーションが DES3 を使って TGT を取得していること。前述のコード例の中に、`csf_gss_acq_user()` を使った例が含まれています。受け入れ側は DES3 の鍵テーブル・エントリを持っている必要があります。
- 開始側アプリケーションが、セキュリティ・コンテキストの開始時に DES3 フラグを使用していること。

必ずしも HP エクステンションを使って DES3 を有効にする必要はありません。ただし、要求された暗号化が使われているかどうかを常に確認しなければなりません。上記の条件を満たしていない場合、コンテキストは、DES3 から DES ヘレベルが下がることがあります。コンテキストを確立した後で、`csf_gss_get_context_options()` を使って戻されたフラグを調べて、DES と DES3 のどちらが使用されたかを確認してください。

## C.10 GSS-API 関数のステータス・コード

GSS-API 関数には、メジャー・ステータスとマイナー・ステータスの 2 つのリターン・ステータス・コードがあります。アプリケーションは、ステータス・コードを使って GSS-API から戻されるエラーを、効率よく処理することが重要です。

メジャー・ステータスは、GSS-API 関数のエラー・コードです。メジャー・ステータス・コードは、ルーチン (関数) エラー、呼び出しエラー、補足情報の 3 つのビット・フィールドで構成されます。

マイナー・ステータスはメカニズムに固有のエラー・コードです。`GSS_ERROR()` によって Kerberos 5 メカニズムに関するエラーが示されている場合に、追加のエラー情報が提供されることがあります。

エラー・コードがあるかどうかは、メジャー・ステータスを `GSS_ERROR()` マクロに送り、得られた結果を検査することで分かります。ゼロが戻された場合、エラーはありません。その他の値が戻された場合は、エラーを示しています。

---

### 警告

---

エラーの有無は、`GSS_ERROR()` マクロを使って調べる必要があります。ステータスを数字のゼロと比較しないでください。

---

`GSS_ERROR()` マクロは、ルーチン (関数) エラー・フィールド、呼び出しエラー・フィールド、または補足情報フィールドエラーがあるかどうかを調べることによってエラーを示します。次のコードは、この使い方の例を示しています。

```
/******  
OM_uint32      status_value = <--- value passed in to test  
OM_uint32      majErr, minErr = 0;  
OM_uint32      message_context = 0;  
gss_buffer_desc status_string = GSS_C_EMPTY_BUFFER;  
  
if ( GSS_ERROR(status_value) ||  
    GSS_SUPPLEMENTARY_INFO(status_value) ) {  
  
    /* First process the Major status code */  
    do {  
        /* Get the status string associated  
         * with the Major (GSS-API) status code */  
        majErr = gss_display_status( &minErr,  
                                    status_value,  
                                    GSS_C_GSS_CODE,  
                                    GSS_C_NO_OID,  
                                    &message_context,  
                                    &status_string );  
  
        /* Print the status string */  
        printf( "Major status string: %s\n",  
                (char*)status_string.value );  
  
        /* Free the status string buffer */  
        gss_release_buffer( &minErr, &status_string );  
    } while( message_context && !GSS_ERROR( majErr ) );  
  
    /* Then process the Minor status code */  
    do {  
        /* Get the status string associated  
         * with the Minor (mechanism) status code */  
        majErr = gss_display_status( &minErr,  
                                    status_value,  
                                    GSS_C_MECH_CODE,  
                                    GSS_C_NO_OID,  
                                    &message_context,  
                                    &status_string );  
    } while( message_context && !GSS_ERROR( majErr ) );  
}
```

```

        /* Print the status string */
        printf( "Minor status string: %s\n",
            (char*)status_string.value );

        /* Free the status string buffer */
        gss_release_buffer( &minErr, &status_string );
    } while( message_context && !GSS_ERROR( majErr ) );
}

/*****

```

補足情報フィールドには、関数の追加情報が提供されます。たとえば、`gss_init_sec_context()` 関数がエラーを示すメジャー・ステータス・コードを戻した場合は、補足情報フィールドに、`gss_init_sec_context()` にアプリケーションのもう一方からのレスポンス・トークンが必要であると示されていることがあります。別の例では、関数 `gss_unwrap()` がメジャー・ステータス・コードを戻した場合、補足情報フィールドには、エラーの原因が順序外のトークンであるか、トークンのリプレイであるか、または古いトークンであることを示しています。

補足情報の値は、メジャー・ステータス・コード名前付きの識別子の AND をとることによって検査します。下記のコード例は、`gss_unwrap()` がリプレイを検出した場合に発生する処理を示しています。

```

/*****
major_status = gss_unwrap(. . .);
if (GSS_ERROR(major_status)) {
    /* An error has occurred. */
    . . .
    if (major_status & GSS_S_DUPLICATE_TOKEN) {
        /* The token is a replay of a previous token. */
        printf ("Warning: Replay detected\n");
    }
}

*****/

```

## C.10.1 マイナー・ステータス・コード

Kerberos メカニズムのマイナー・エラー・コードは、`include/csf/sts` および `include/csfc5/sts` ディレクトリの `.hs` ファイルにあります。

これらのエラー・コードはどれも、アプリケーションから参照でき、その原因となった Kerberos のエラー状況に応じてコードを分岐させることもできます。アプリケーションは、値を定義するマクロを参照する必要があります (`#define` の後のシンボル)。

通常、マイナー・エラーは、Kerberos の構成、またはプリンシパル・データベースのプリンシパルに問題があることを示します。

## C.11 サンプル・プログラム

Application Security SDK には、サンプルのクライアント/サーバ・アプリケーションが含まれています。プログラムは、GSS-API 関数の使用例を示します。サンプル・プログラムでは、下記の GSS-API 関数を使っています。

- `gss_accept_sec_context()`
- `gss_acquire_cred()`
- `gss_delete_sec_context()`
- `gss_display_name()`
- `gss_display_status()`
- `gss_export_sec_context()`
- `gss_get_mic()`
- `gss_import_name()`
- `gss_import_sec_context()`
- `gss_init_sec_context()`
- `gss_inquire_context()`
- `gss_inquire_names_for_mech()`
- `gss_oid_to_str()`
- `gss_release_buffer()`
- `gss_release_cred()`
- `gss_release_name()`
- `gss_release_oid()`
- `gss_release_oid_set()`
- `gss_str_to_oid()`
- `gss_unwrap()`
- `gss_verify_mic()`
- `gss_wrap()`



クライアント・プログラムは、サーバ・プログラムとのセキュリティ・コンテキストを開始します。セキュリティ・コンテキストが確立された後、クライアントとサーバは、保護されたメッセージを交換します。クライアント・プログラムとサーバ・プログラムは、同じレルム内の 2 つのシステム、または (レルム間認証の設定がなされている) 異なるレルムの 2 つのシステムに分散できます。

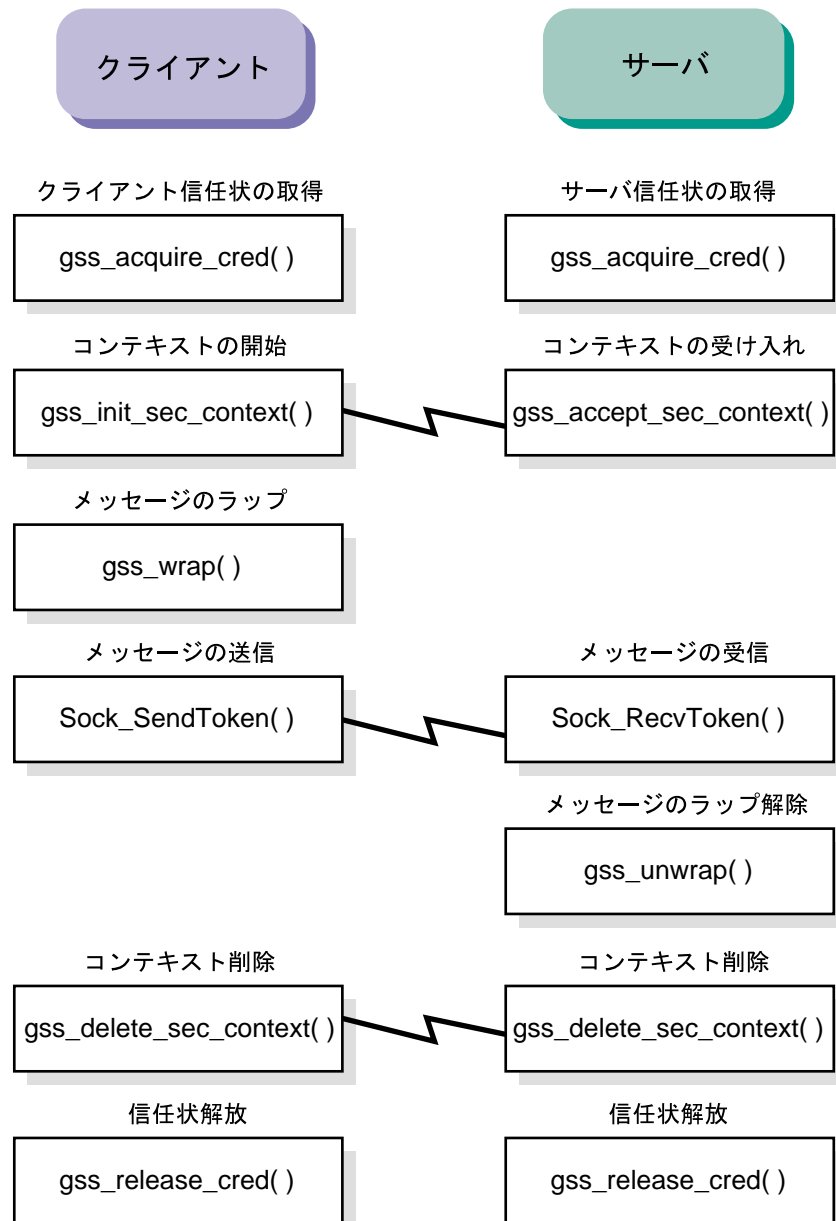
クライアントは、呼び出されるたびにサーバとのやり取りを 1 回または複数回実行します。サーバとの 1 回のやり取りは、主に次の手順で行われます。

1. TCP/IP 接続が確立されます。
2. (オプションで、省略時には) クライアントとサーバは GSS-API コンテキストを確立し、サーバはクライアントの識別情報を表示します。
3. クライアントがサーバにメッセージを送信します。メッセージは、プレーン・テキスト、暗号処理によって「署名」された暗号化されていない形式、または暗号化された形式 (省略時の設定) のいずれかになります。
4. サーバは、(必要であれば) メッセージを復号し、(もしあれば) 署名をチェックし、メッセージを表示します。
5. サーバは、メッセージの受け入れ確認のために、署名ブロック (省略時の場合) または空のトークンをクライアントに戻します。
6. サーバが署名ブロックを送信すると、クライアントはこれをチェックし、チェックが終了したことを示すメッセージを表示します。
7. クライアントは、サーバに空のブロックを送信して、やり取りが終了したことを伝えます。
8. クライアントとサーバは、TCP/IP 接続を閉じて、GSS-API コンテキストを破壊します。

ソース・コード	説明
<code>gss-server.c</code>	メインのサーバ・ソース・コード。
<code>gss-client.c</code>	メインのクライアント・ソース・コード。
<code>gss-misc.h</code>	その他のタスクのヘッダ・ファイル。
<code>gss-misc.c</code>	その他のタスクを実行します。

ソース・コード	説明
Makefile_CPQ	ビルド・ファイル。
README	情報ファイル。

次のダイアグラムは、GSS 関数がサンプル・プログラムの中でどのように実装されているかを示しています。



ZK-1834U-AI

### C.11.1 サンプル・プログラムのビルド

HP では、サンプル・コードを別の場所にコピーして、変更が必要になった場合はオリジナルのコードを保存しておくことを推奨しています。提供されている makefile (Makefile\_CPQ) は、ほとんど、あるいはまったく変更する必要がありません。省略時の場合、デバッグは無効です。

実行可能プログラムをビルドするには、次の make コマンドを使います。

- 省略時の設定で実行可能プログラムをビルドするには、次のように入力します。

```
# make -f Makefile_CPQ
```

- デバッグを有効にするには、次のように入力します。

```
# make -f Makefile_CPQ OPTDEB=-g2
```

- オブジェクトと実行可能プログラムを削除するには、次のように入力します。

```
# make -f Makefile_CPQ clean
```

### C.11.2 サンプル・プログラムの実行

ここでは、以下について説明します。

- サンプル・プログラムの実行前に満たすべき前提条件
- サンプル・プログラムの開始方法
- サンプルのサーバ・プログラムのコマンド
- サンプルのクライアント・プログラムのコマンド
- 標準的なサンプル・プログラムの出力
- トラブルシューティングのガイドライン

#### C.11.2.1 前提条件

サンプル・プログラムを実行するには、次の要件を満たしておく必要があります。

1. クライアントのプリンシパルを定義する。

クライアント・プログラムは、プリンシパルの `host/host_name@REALM` を使って認証します (`host_name` は受け入れ側ホストの完全修飾ドメイン名、`REALM` は受け入れ側ホストの省略時のレルムです。したがっ

て、プリンシパル `host/host_name@REALM` はセキュリティ・サーバ (KDC) に登録されていなければなりません。

2. サーバ・プリンシパルを取り出す。

サーバのホスト・プリンシパルがサービス鍵テーブル・ファイル (`v5srvtab`) にあり、サービス鍵テーブル・ファイルを、サーバ・プログラムから読み取ることができなければなりません。サービス鍵テーブル・ファイルがセキュリティ・サーバにない場合は、ホスト・プリンシパルを取り出すことで作成できます。

3. クライアントのプリンシパルを、プリンシパル・データベースに追加する。

ホストでサーバを実行するには、`service_name` に対応するプリンシパルが、サーバ・ホストの省略時の鍵テーブル (`/krb5/v5srvtab`) にあり、`gss-server` プロセスがこの鍵テーブルを読み取ることができなければなりません。たとえば、サービス名 “`sample@server`” は、Kerberos のプリンシパル “`sample/KDChost.domain.com@REALM`” に対応します。

### C.11.2.2 サンプル・プログラムの設定

必ず、サンプル・サーバ・プログラムを先に実行し、その後サンプル・クライアント・プログラムを実行してください。サンプルの出力を別々に確認できるように、各プログラムを別々のシェルで実行します。クライアントとサーバは、同じマシン実行することも、別々のマシンで実行することもできます。

クライアント/サーバ・プログラムは、必須のコマンド行引数を受け取ります。スイッチはすべてオプションです。

サーバ・プログラムには次の引数が必要です。

- `service_name`— “`service@host`” の形式の GSS-API サービス名。

たとえば次のとおりです。

```
# gss-server sample@REALM
```

クライアント・プログラムには次の引数が必要です。

`host`                      サーバを実行するホスト。

`service_name`            サーバが接続の確立に使うサービス名。

msg

複数の単語の場合は、次のようにメッセージを二重引用符で囲みます。

以下に、サーバのコマンド用スイッチについて説明します。スイッチはすべて省略可能です。

<code>-port</code>	接続を受け入れる TCP ポート。
<code>-once</code>	1 回のやり取りが終わったら、継続せずに終了するようにサーバに指定します。
<code>-inetd</code>	<code>inetd</code> を経由せずに実行して、ネットワーク・ポートにバインドせずに、 <code>stdin</code> のクライアントとやり取りするようにサーバに指定します。 <code>-once</code> が暗黙のうちに指定されます。
<code>-export</code>	クライアントとのコンテキストの確立後、 <code>gss_export_sec_context()</code> 関数をテストするようサーバに指定します。
<code>-logfile</code>	サーバの出力を <code>stdout</code> ではなくファイルに送るときの出力先ファイル。

以下に、クライアントのコマンド用スイッチについて説明します。スイッチはすべて省略可能です。

-d	信任状をサーバに委任するようクライアントに指定します。Kerberos GSS-API メカニズムの場合、これは、転送可能 TGT がサーバに送信され、サーバの信任状キャッシュに格納されることを意味します (これを動作させるには、 <code>kinit -f</code> でチケットを取得する必要があります)。
-f	<code>msg</code> 引数が、実際にはファイルの名前であり、その内容をメッセージとして使う必要があることをクライアントに指定します。
-q	表示を抑制して、エラー・メッセージだけを表示するようにクライアントに指定します。
-ccount	クライアントがサーバと開始する必要があるセッションの数 (「接続数」) を指定します。
-mcount	各セッションでサーバに送信すべきメッセージの回数 (「メッセージ数」) を指定します。
-na	サーバとの認証をまったく行わないようクライアントに指定します。-nw, -nx, -nm が暗黙のうちに指定されます。
-nw	メッセージを「ラップ」しないようクライアントに指定します。-nx が暗黙のうちに指定されます。
-nx	メッセージを暗号化しないようクライアントに指定します。
-nm	暗号化を使ったチェックサム (「MIC」) を戻すことをサーバに要求しないようクライアントに指定します。

引数が指定されていない場合やプログラムが正しく使われていない場合は、クライアント/サーバ・サンプルの使用法のメッセージが表示されます。

```
# gss-server
Usage: gss-server [-port port] [-verbose] [-once]
        [-inetd] [-export] [-logfile file] [service_name]

# gss-client
Usage: gss-client [-port port] [-mech mechanism] [-d]
        [-f] [-q] [-ccount count] [-mcount count]
        [-na] [-nw] [-nx] [-nm] host service msg
```

### C.11.3 サンプル・プログラムの出力

ここでは、サンプルのクライアント・サーバ・プログラムの出力を取り上げます。ここでは、クライアントからはメッセージとしてテキスト・ファイルを渡し、サーバはそのクライアントを認証してメッセージを表示しています。

```
# gss-server -verbose -once sample@REALM
Server waiting:
Received token (size=600):
60 82 02 54 06 05 2b 05 01 05 02 01 00 6e 82 02
```

```
47 30 82 02 43 a0 03 02 01 05 a1 03 02 01 0e a2
07 03 05 00 20 00 00 00 a3 82 01 56 61 82 01 52
30 82 01 4e a0 03 02 01 05 a1 15 1b 13 43 59 42
45 52 4e 54 2e 5a 4b 33 2e 44 45 43 2e 43 4f 4d
a2 28 30 26 a0 03 02 01 03 a1 1f 30 1d 1b 06 73
61 6d 70 6c 65 1b 13 63 79 62 65 72 6e 74 2e 7a
6b 33 2e 64 65 63 2e 63 6f 6d a3 82 01 04 30 82
01 00 a0 03 02 01 05 a1 03 02 01 01 a2 81 f3 04
81 f0 8e 80 d6 1d b4 11 51 61 75 24 f4 e4 96 c5
db 1b ba 94 be c1 60 11 1a 24 c3 13 ae 22 50 90
e2 ba 18 7b 9e 8f 6a 42 ca fa 96 b8 cf f0 8c 3f
b8 ea 33 e9 21 24 ab ab e2 7e 4a 90 ee 75 93 11
99 4e 15 ad 2e 47 83 5b de 74 eb b7 92 ad 86 e6
27 fd c0 02 13 6d 56 38 3d 7e 80 dc ea a0 1c 79
37 34 6f fa 4b dc 79 ed 2b ee ef 93 37 ae 6c 1f
2c 83 62 c1 7a 0b 5a aa 10 47 e4 70 1d 31 9c 2c
24 ee 8e 69 4a e2 c2 cd 7e 52 d4 ff 19 d6 d4 77
e9 ee 78 19 e8 2d 5b 31 5c a0 89 28 f5 b2 e2 ef
bf 2a 12 b5 1b 88 e3 e8 a3 21 5b d5 93 a1 21 44
77 e8 e1 71 f3 2f b8 e1 06 20 fe 21 42 9a f5 bf
e3 7b 55 76 a2 54 34 05 43 14 0b aa 2e d2 da 91
31 02 8d 65 53 fe 4d 32 b6 b6 31 6f b2 7c d4 77
bb 87 0f 85 6d 61 8d 2c 21 e8 be 3d fb fd a7 72
bd 71 a4 81 d3 30 81 d0 a0 03 02 01 05 a1 03 02
01 00 a2 81 c3 04 81 c0 aa b1 ae d7 0a 9b 75 c9
ce ca b4 b1 1d a4 a4 4b bc 3a 73 4d b8 9e c0 fb
51 44 8a 67 8d ad 25 87 6e 66 ed bf d7 fb fc b1
6b 38 89 74 2b f8 eb 04 be 76 70 03 e3 2f db 7d
15 53 53 9e 8d e4 f1 a1 60 b6 01 33 42 90 60 a5
4c 75 dd af af 64 75 86 5a f8 25 57 c1 22 bc 12
b1 54 c9 c1 0b a6 27 c0 44 2a 84 48 a6 6a 00 62
f8 4d d4 df 6f 6e 19 99 99 68 64 78 53 27 22 8a
d9 52 84 40 5b 19 fe 4d bf eb 07 d0 e0 f9 46 8c
72 5b bc df d0 0e 55 4a e2 e8 39 10 83 63 9f 02
cd 07 8b 00 f9 d6 46 77 29 6d 13 64 e4 3c b3 53
f7 46 12 9c 88 8b 5d e6 9e 05 f2 f7 6d 4e d8 15
dd eb 9b a6 37 28 77 9a
```

Sending accept\_sec\_context token (size=107):

```
60 69 06 05 2b 05 01 05 02 02 00 6f 5e 30 5c a0
03 02 01 05 a1 03 02 01 0f a2 50 30 4e a0 03 02
01 05 a1 03 02 01 00 a2 42 04 40 b1 61 8f 22 03
93 d2 d4 df 6f 6e 19 99 99 68 64 78 53 27 22 8a
c4 e8 42 b3 b5 23 3f 5d 0d 6a 6d 87 42 7b ca 8b
10 13 ff 34 66 23 cf 75 a0 24 e9 4a a7 d6 cd 9c
e2 a5 1b 98 55 02 ba 85 e8 3c b3
```

context flag:GSS\_C\_MUTUAL\_FLAG

context flag:GSS\_C\_REPLAY\_FLAG

context flag:GSS\_C\_CONF\_FLAG

context flag:GSS\_C\_INTEG\_FLAG

Accepted connection using mechanism OID { 1 2 840 113554 1 2 2 }.



```

Accepted connection: "user@REALM"
Message token (flags=228):
60 57 06 05 2b 05 01 05 02 02 01 dd fa de fa ff
ff e7 ad 91 9a ca e2 08 aa 32 27 86 fa 8a 0c 17
44 18 9c 8c 7b f4 65 8c 63 88 6f be 70 f0 a8 ef
95 17 da 92 48 44 e6 70 1c 4a 80 97 c0 f3 d3 34
39 1f 03 b3 55 df 50 75 f0 40 9d 8a 9b 5d 0f aa
3d 6b a0 c6 6d 34 42 29 58
Received message: "
Hello Kerberos.
I am a text file.
"
NOOP token
#

```

以下は、クライアントからの出力を `test` というテキスト・ファイルに送っています。

```

# gss-client -f unix1 sample@REALM "test"
Sending init_sec_context token (size=600)...continue needed...
context flag:GSS_C_MUTUAL_FLAG
context flag:GSS_C_REPLAY_FLAG
context flag:GSS_C_CONF_FLAG
context flag:GSS_C_INTEG_FLAG
"user@REALM" to "sample/host.domain@REALM", lifetime -1,
flags 1b6, locally initiated, open
Name type of source name is { 1 2 840 113554 1 2 2 1 }.
Mechanism { 1 2 840 113554 1 2 2 } supports 7 names
  0: { 1 2 840 113554 1 2 1 1 }
  1: { 1 2 840 113554 1 2 1 2 }
  2: { 1 2 840 113554 1 2 1 3 }
  3: { 1 2 840 113554 1 2 1 4 }
  4: { 0 18 18 18 18 18 18 18 18 18 18 }
  5: { 1 2 840 113554 1 2 2 1 }
  6: { 1 2 840 113554 1 2 2 2 }
Signature verified.
#

```

#### C.11.4 トラブルシューティングのガイドライン

サンプル・プログラムの実行に関する問題のほとんどは、セキュリティ・サーバのセットアップと管理に関連しています。しかし、サンプル・プログラムを正しく実行できない場合は、次のようにいくつか確認すべき項目があります。

- 一般に、サーバ・プログラムは `root` 以外のユーザが実行します。しかし、省略時の場合、サービス鍵テーブル・ファイルは `root` のみが読み取り可能です。

サーバ・プログラムがサービス鍵テーブル・ファイルを読み取れることを確認し、さらにこのファイルに `sample/KDC_host_name@REALM` のエントリが格納されていることを確認してください。

- Application Security SDK 用の Kerberos ライブラリは、サービス鍵テーブル・ファイルの場所を指定する 2 つの環境変数をサポートしています。すなわち、`CSFC5KTNNAME` および `KRB5KTNNAME` です。両方の環境変数が設定されている場合は前者が優先します。この 2 つの変数がサービス鍵テーブル・ファイルの場所を正しく指定しているかどうかを確認し、さらに有効な TGT (`klist`) を持っていることを確認してください。
- サンプル・プログラムは、レルム認証のための設定がなされている場合に限り、複数のレルムをまたがって動作します。

Application Security SDK は Kerberos メカニズムの上層に位置しています。このため、何か不具合が生じた場合、通常は基盤のメカニズムに支障が生じています。サンプル・プログラムは単にエラーを渡しているだけです。たとえば、間違ったパスワードが入力された場合、Kerberos は “Decrypt integrity check failed (復号完全性チェックに失敗しました)” といったメッセージを生成し、これを Application Security SDK に送信します。この場合、サンプル・プログラムは単にエラーを報告し、後始末をして終了します。

**A****A\_PROCMASK\_SET** マクロ ... 5-7**ACCEPT**

省略時の名前 ..... 8-39

**ACL**

umask ..... 7-9

エントリ規則 ..... 7-9

オブジェクトの作成規則 ..... 7-9

外部表現 ..... 7-5

継承 ..... 7-14

作業域 ..... 7-2

作業域例 ..... 7-10

省略時 ..... 7-14

引き継ぎ ..... 7-9

ファイルの設定の例 ..... 7-10

複製規則 ..... 7-9

ライブラリ・ルーチン ..... 7-8

**allowSendEvents** リソース .... 2-8**ANSI C**

シンボル優先使用 ..... 6-27

**Application Security SDK** .... 8-5**AUD\_MAXEVENT\_LEN** ..... 5-8**AUD\_TP** プライベート監査トーク

ン ..... 5-4

**AUD\_T** パブリック監査トークン 5-3**audgenl()**

例 ..... 5-8

**audgenl** システム・コール

例 ..... 5-9

**audgenl** ライブラリ・ルーチン

例 ..... 5-1

**audgen** システム・コール ..... 5-1

監査ログの指定 ..... 5-10

**C****chown** システム・コール

SUID 許可または SGID 許可 .. 2-1

**close-on-exec** フラグ ..... 2-7**create\_file\_securely()** ライブラリ・

ルーチン ..... 3-3

**D****DAC**

TCB の保護 ..... 1-4

**DES-CBC** ..... 8-24**DES-MAC** ..... 8-24**DES-MAC-MD5** ..... 8-23**DES3** ..... C-38**DES3-CBC** ..... 8-24**DES3-MAC-MD5** ..... 8-23**E****EACCES** **errno** 値 ..... 2-3**EPERM** **errno** 値 ..... 2-3

**EROFS errno** 値 ..... 2-3  
**errno** 変数 ..... 2-3  
**/etc/auth/system/ttys** ファイル 3-6  
**/etc/passwd** ファイル ..... 3-4, 4-5  
**/etc/sec/audit\_events** ファイル B-1  
**/etc/sec/event\_aliases** ..... B-5  
**/etc/sec/site\_events** ファイル .. 5-8  
**/etc/sysconfigtab**  
    サイト定義監査イベントの設定 5-8  
**execve** システム・コール ..... 2-6

## F

**fcntl** システム・コール  
    close-on-exec フラグ ..... 2-7  
**fork** システム・コール ..... 2-6, 4-1

## G

**getluid** システム・コール ..... 4-2  
**gss\_accept\_sec\_context** 関数 8-18,  
    8-19  
    チャンネル・バインディング... 8-19  
**gss\_acquire\_cred** 関数  
    OID の受け渡し ..... 8-37  
**GSS-API**  
    主な目的 ..... 8-1  
    コンテキスト管理 ..... 8-16  
    使用される名前 ..... 8-7  
    セキュリティ・メカニズム... 8-2  
    設計上の目標 ..... 8-1  
    前提条件 ..... 8-3  
    トランスポート・プロトコルからの  
        独立 ..... 8-1  
    標準 ..... 8-4  
    メカニズムからの独立 ..... 8-1

### GSS\_C\_AF\_INET

    サポートされているアドレス・  
        フォーマット ..... 8-19

**GSS\_CALLING\_ERROR** 関数 8-50

**gss\_compare\_name** 関数 ..... 8-7

**gss\_display\_name** 関数

    名前の比較 ..... 8-36

    表示可能な名前 ..... 8-36

**GSS\_ERROR** 関数 ..... 8-50

**gss\_get\_mic** 関数

    QOP の指定 ..... 8-37

**gss\_import\_name** 関数

    名前の解析 ..... 8-10

    名前の比較 ..... 8-36

    表示可能な名前 ..... 8-37

**gss\_init\_sec\_context** 関数

    OID の受け渡し ..... 8-37

    optional protection

        完全性 ..... 8-20

    オプションの保護

        機密性 ..... 8-20

        順序外メッセージ検出 ..... 8-21

        チケットの転送 ..... 8-21

        チャンネル・バインディング 8-18

        リプレイ検出 ..... 8-20

    開始側の責任 ..... 8-21

    チャンネル・バインディング... 8-20

    メカニズム識別子 ..... 8-17

**gss\_release\_name** 関数

    使用のタイミング ..... 8-9

**GSS\_ROUTINE\_ERROR** 関数 8-51

**gss\_seal** 関数 ..... 8-28t

    (**gss\_wrap** 関数 も参照)

**gss\_sign** 関数 ..... 8-28t

    (**gss\_get\_mic** 関数 も参照)

## **GSS\_SUPPLEMENTARY\_INFO**

- function** ..... 8-52
- gss\_unseal** 関数 ..... 8-28t  
( gss\_unwrap 関数 も参照 )
- gss\_unwrap** 関数  
戻される QOP ..... 8-38
- gss\_verify\_mic** 関数  
戻される QOP ..... 8-38
- gss\_verify** 関数 ..... 8-28t  
( gss\_verify\_mic 関数 も参照 )
- gss\_wrap** 関数  
QOP の指定 ..... 8-37

## **I**

### **INITIATE**

- 省略時の名前 ..... 8-38

### **Internet Drafts**

- Generic Security Service API  
Version 2: C-bindings ..... 8-4

### **iovec**

- 監査レコードの使用 ..... 5-4

## **K**

### **Kerberos**

- 概要 ..... C-1

### **Kerberos** 固有のエラー・コード

- ..... 8-57

### **Key Distribution Center**

- (KDC) ..... C-3

## **L**

- libaud** ライブラリ ..... 1-5
- libsecurity** ライブラリ ..... 1-5

## **M**

- major status** ..... 8-53
- matrix.conf** ファイル ..... 6-25
- MD2.5** ..... 8-23
- MIN\_SITE\_EVENT** ..... 5-8

## **O**

- OID** ..... 8-17, 8-25t  
使用 ..... 8-9
- OID** セット  
ハード・コードされた ..... 8-37

## **P**

### **PATH** 変数

- 安全なシェル・スクリプト... 2-10
- 定義 ..... 2-5
- ヌル・エントリ ..... 2-5
- Pretty Good Privacy (PGP)**.. 8-23

## **Q**

### **QOP**

- Kerberos ..... 8-23
- 指定 ..... 8-37
- 使用 ..... 8-22
- 省略時の使用 ..... 8-38

### **quality of protection**

( QOP を参照 )

## R

### RFC

- 1510 Kerberos Network Authentication Service (V5) 8-4
- 1964 Kerberos Version 5 GSS-API Mechanism ..... 8-4
- 2078 Generic Security Service Application Program Interface Version 2..... 8-4

## S

### [Secure Keyboard] メニュー項

- 目..... 2-7

### set\_auth\_parameters() ライブラ

- リ・ルーチン ..... 4-2

### setluid システム・コール..... 4-2

### SGID

- セット・グループ ID プログラム ..... 2-1

### SIA

- finger 情報の変更 ..... 6-22
- group 情報 ..... 6-24
- login プロセス ..... 6-21
- matrix.conf ファイル..... 6-25
- password へのアクセス ..... 6-23
- rlogind プロセス ..... 6-21
- rshd プロセス..... 6-21
- SIAENTITY 構造体 ..... 6-8
- siainit コマンド..... 6-7
- sialog ファイル ..... 6-11
- インタフェース・ルーチン.... 6-3
- 階層型 ..... 6-6

- 監査ログ..... 6-12
- コーディング例 ..... A-1
- コールバック ..... 6-8
- 状態の保守..... 6-10
- 初期設定..... 6-7
- セキュア情報の変更..... 6-21
- セキュア情報へのアクセス... 6-22
- セキュリティを扱うコマンド . 6-2
- セッション処理 ..... 6-14
- セッションの解放..... 6-20
- セッションの確立..... 6-20
- セッションの起動..... 6-20
- セッションの初期化..... 6-18
- セッションの認証..... 6-19
- デバッグ..... 6-12
- パスワードの変更..... 6-21
- パラメータの収集..... 6-8, 6-24
- ヘッダ・ファイル..... 6-7
- 保証 ..... 6-13
- メカニズムに依存するインタフェース ..... 6-25
- メカニズムの統合..... 6-12
- ユーザのシェルの変更 ..... 6-22
- リターン値..... 6-11, 6-14
- レイヤード・プロダクトのパッケージ ..... 6-24
- ログ ..... 6-11
- signal** ルーチン ..... 2-6
- SIGQUIT** シグナル  
セキュリティ上の考慮事項.... 2-6
- SIGTRAP** シグナル  
セキュリティ上の考慮事項.... 2-6
- site\_events** ファイル ..... 5-8
- SUID**  
実行可能スタック..... 2-2

セット・ユーザ ID プログラム 2-1

## T

**TCB**..... 1-2  
間接プログラム ..... 1-2  
実行可能ファイル..... 1-3  
セキュリティ構成 ..... 1-5  
トラステッド・システム・ディレク  
トリ ..... 1-6  
トラステッド・プログラム.... 1-2  
**tmp** ファイル  
セキュリティ上の考慮事項.... 2-5

## U

**umask** システム・コール ..... 7-9  
一時ファイルの保護..... 2-4  
**unlink** システム・コール  
ファイル・アクセスの保護.... 2-4  
**/usr/tmp** ファイル ..... 2-5

## V

**v5srvtab** ファイル..... C-14

## X

**XGrabKeyboard()** ルーチン.... 2-7  
**XReparentWindow()** ルーチン  
セキュアな環境での使用 ..... 2-9  
**XSendEvent()** ルーチン ..... 2-8  
**X** ウィンドウ  
(**X** 環境 を参照)  
**X** 環境

安全なプログラムの書き方.... 2-7

セキュアな環境での使用 ..... 2-7

## あ

アカウントのロック ..... 4-4  
アクセス制御リスト ..... 8-7  
暗号化  
DES3 の使用..... C-38  
アルゴリズム ..... C-24  
タイプ ..... 8-29  
暗号化されたパスワード ..... 3-4

## い

一時ファイル..... 2-4, 3-3  
イベント  
エイリアス..... B-5  
監査 ..... 5-2, B-1

## う

受け入れ側 ..... C-9

## え

永続ファイル..... 2-4  
エクスポート  
セキュリティ・コンテキスト 8-29  
名前 ..... 8-7  
エラー・コード ..... C-39  
Kerberos 固有..... 8-57  
エラー処理マクロ ..... 8-49  
エントリ・ポイント ..... 6-27

エンハンスド・パスワード・データベース ..... 3-4, 4-3

## お

オブジェクト・コード ..... 1-3  
オブジェクト識別子  
（OID を参照）  
親ディレクトリ ..... 1-3  
オープン・ファイル記述子 ..... 2-6

## か

開始側 ..... C-9  
外部表現  
ACL ..... 7-5  
外部名 ..... 8-7  
鍵  
秘密 ..... C-4  
空のパスワード ..... 4-4  
環境変数  
CSFC5CCNAME ..... 8-15  
CSFC5KTNAME ..... 8-15, C-52  
CSFC5RCNAME ..... 8-33  
KRB5KTNAME ..... C-52  
監査  
AUD\_TP プライベート・トークン ..... 5-4  
AUD\_T パブリック・トークン ..... 5-3  
audctl フラグ ..... 5-6  
auditmask フラグ ..... 5-6  
iovec タイプ・トークン ..... 5-3  
アプリケーション固有のレコード ..... 5-7  
イベント・タイプ ..... 5-2  
固定長トークン ..... 5-3

サイト定義イベント ..... 5-8  
システム・コール監査の無効化 ..... 5-6  
タプル ..... 5-2  
独自のログの作成 ..... 5-10  
トークン ..... 5-2  
プロセス制御フラグ ..... 5-4  
プロセスの変更 ..... 5-6  
ポインタ・タイプ・トークン ..... 5-3  
マスク ..... 5-4  
連続したタプルとしてのレコード ..... 5-2

### 監査イベント

省略時のイベント ..... B-1  
監査イベント・エイリアス ..... B-5  
監査可能なイベント ..... B-1  
監査サブシステム

省略時のイベント・エイリアス ..... B-5  
省略時の監査可能なイベント ..... B-1

### 監査ログ

タプルのフォーマット ..... 5-11  
読み取り ..... 5-10  
読み取りのアルゴリズム ..... 5-16

### 関数

基本タスクの実行 ..... C-8  
セキュリティ・コンテキスト管理 ..... 8-16  
その他のサポート ..... 8-24  
名前管理 ..... 8-6  
リファレンス ..... 8-39  
完全性 ..... 8-1, 8-20, C-2  
アルゴリズム ..... C-24

## き

機密性 ..... 8-1, 8-20, C-2



キャッシュ	
管理 .....	8-29
省略時の位置 .....	8-15
キーボード	
セキュア .....	2-8

## く

クライアント .....	C-2
クロック・スキュー .....	C-38

## こ

コア・ファイル .....	2-6
子プロセス	
継承されるファイル・アクセス .....	2-6
シグナル・マスク .....	2-6
コンテキスト	
確立 .....	C-16
終了 .....	C-33

## さ

最初の決定事項 .....	C-7
サイト定義監査イベント .....	5-8
作業域	
ACL .....	7-2
サンプル・アプリケーション ..	C-42
サンプル・プログラム	
実行 .....	C-46
前提条件 .....	C-46
名前のインポート .....	C-11
名前の表示 .....	C-11
ビルド .....	C-46

サービス .....	C-2, C-3
サービス鍵テーブル・ファイル ..	8-33
省略時の名前 .....	C-14
信任状の格納 .....	C-14
サービス・チケット .....	C-5
サービス・プリンシパル名 ....	C-11

## し

シェル	
パス変数の構文 .....	2-5
変数の定義 .....	2-5
シェル・スクリプト .....	1-3
セキュリティ上の考慮事項 ...	2-10
シェル変数	
特定のシェル変数 .....	2-5
識別 .....	4-1, C-2
シグナル	
安全な応答 .....	2-5
時刻同期 .....	C-38
システム・コール	
コマンドの戻り値 .....	2-2
失敗したシステム・コールに関する	
セキュリティ上の考察 .....	2-3
システム省略時設定データベース	
説明 .....	3-3
未定義フィールド .....	3-6
事前認証 .....	8-34
実行可能スタック .....	2-2
承認 .....	C-2
省略時の名前 .....	8-38
( 名前 も参照 )	
省略時のプリンシパル .....	8-9
初回信任状	

管理 .....	C-36
信任状	
Kerberos の省略時のキャッ	
シュ .....	8-15
委任 .....	8-21
受け入れ側.....	C-14
開始側 .....	C-14
管理 .....	8-10
期限切れ.....	8-14
取得 .....	C-12
ユーザ.....	C-34
初回 .....	C-4
管理.....	C-36
タイプ .....	C-13
リフレッシュ .....	8-33
信任状のリフレッシュ.....	8-33
シンボル優先使用 .....	6-27

## す

スタック	
実行可能 .....	2-2
スタートアップ・スクリプト ....	4-1
スティッキ・ビット .....	1-4
一時ファイルの保護.....	2-4
ステータス・コード	
リターン値.....	C-39
スレッド .....	8-28

## せ

セキュリティ	
考慮事項 .....	8-28
推奨事項 .....	8-28
メカニズム.....	8-2
セキュリティ・コンテキスト	

オプションのセキュリティ..	C-17
機密性と完全性.....	8-20
使用可能 .....	8-18
チャンネル・バインディング	8-18
メッセージ順序.....	8-21
リプレイ検出.....	8-20
確立 .....	C-16
終了 .....	C-33
セキュリティ侵害	
考えられるプログラムの応答 .	2-3
セキュリティ・メカニズムの統	
合.....	6-12
セキュリティを扱うコマンド ....	6-2
セグメント.....	2-4
絶対パス名 .....	2-5
セマフォ .....	2-4
前提条件 .....	8-3

## そ

相互認証 .....	8-21, 8-32, C-7
相対パス名 .....	2-5

## た

タブル	
監査の解析.....	5-16
監査ログ共通 .....	5-11
詳細説明.....	5-12
端末制御データベース.....	3-5, 3-7

## ち

遅延.....	3-5
チケット .....	C-3
延長期限.....	8-35

サービス ..... C-5  
事前認証 ..... 8-34  
初回 ..... C-4  
属性 ..... 8-34  
存続期間 ..... 8-35, C-38  
転送可能 ..... 8-34  
チケット・グランディング・サービス  
(TGS) ..... C-5  
チケット・グランディング・チケット  
(TGT) ..... C-4  
取り出し ..... C-14  
チャンネル・バインディング ..... 8-18  
アプリケーション・データ... 8-18

## つ

強いシンボル ..... 6-28

## て

定数 ..... 8-44  
デバイス  
データベース ..... 3-2  
転送可能チケット ..... 8-21, 8-34  
データ  
安全な位置への格納 ..... 2-4  
データ構造体 ..... 8-47  
データ・ファイル ..... 1-3  
データベース  
アクセス ..... 3-13  
エントリ ..... 3-6  
エントリの書き込み ..... 3-12  
更新 ..... 3-8, 3-12  
システム省略時設定 ..... 3-6

端末制御 ..... 3-7  
フィールド ..... 3-6  
データベース・エントリの変更 ..... 3-12  
デーモン・プログラム ..... 4-2

## と

トラステッド・コンピューティング・  
ベース  
(TCB を参照)  
トラステッド・プログラミング技  
法 ..... 2-1  
トラステッド・プログラム ..... 1-2  
トランスポート・プロトコルからの独  
立 ..... 8-1  
トリプル **DES** ..... C-38  
トークン  
iovec タイプの監査 ..... 5-3  
固定長の監査 ..... 5-3  
パブリックの監査 ..... 5-3  
プライベートの監査 ..... 5-4  
ポインタ・タイプ・トークン ..... 5-3  
トークン・カード ..... 8-5  
ワン・タイム・パスワード.. C-36  
トークンの交換 ..... 8-17, C-17

## な

内部名 ..... 8-7, C-10  
名前  
default  
プリンシパル名 ..... 8-9  
エクスポート ..... 8-7  
外部 ..... 8-7

取得 ..... C-10  
使用 ..... 8-7  
省略時 ..... 8-38, C-11  
    ACCEPT..... 8-39  
    INITIATE..... 8-39  
    サービス・プリンシパル名 C-11  
内部 ..... 8-7, C-10  
判読可能 ..... C-10  
比較 ..... 8-36  
表示可能 ..... 8-36  
メカニズム..... 8-7  
名前タイプ ..... 8-7  
Kerberos 5..... C-11

## に

認証..... 4-1, 8-1, C-2  
    相互 ..... 8-21, 8-32, C-7  
認証サービス..... C-3  
認証子..... C-6  
認証データベースのアクセス ... 3-13  
認証プログラム ..... 4-1  
認証プロファイル ..... 3-4

## は

はじめに ..... C-7  
パス名  
    絶対 ..... 2-5  
    相対 ..... 2-5  
パスワード  
    コーディング例 ..... A-1  
    保護 ..... 8-32  
パスワードの保護 ..... 8-32

バッファ管理..... 3-8

## ひ

表記規則 ..... 8-42  
標準規格の情報 ..... 8-4

## ふ

ファイル  
    保護 ..... 2-4  
ファイル記述子 ..... 2-7  
ファイル制御データベース  
    説明 ..... 3-2  
プライベート監査トークン ..... 5-4  
プリンシパル..... C-3  
    サービス名..... C-11  
    無人運用ホスト ..... C-12  
プリンシパル・データベース .... C-3  
プリンシパル名 ..... 8-9  
プロセス  
    監査制御フラグ ..... 5-4  
プロセス間通信  
    セキュリティ上の考慮事項.... 2-4  
プロセスの優先順位 ..... 3-4

## へ

ヘッダ・ファイル ..... 1-5

## ほ

保護サブシステムの疑似グループ 3-8  
保証..... 6-13

## ま

- マイナー・ステータス..... 8-55,  
C-39, C-41
- マウス
  - セキュア..... 2-8
- マクロ
  - 監査の解析..... 5-16
- マルチ・スレッド関数..... 8-31

## み

- 未定義フィールド ..... 3-6

## め

- 命名, ルーチン ..... 6-28
- メカニズム
  - 指定 ..... 8-37
  - 独立 ..... 8-1
  - 名前 ..... 8-7
- メカニズムに依存するインタフェース..... 6-25
- メジャー・ステータス..... C-39
- メッセージ
  - KDC との交換 ..... C-6
  - アプリケーション間での交換 C-6,  
C-24
  - 完全性 ..... C-24
  - 機密性 ..... C-24
  - 順序 ..... 8-21

## ゆ

- ユーザ・コンテキスト..... C-7
- ユーザの入力
  - セキュリティ上の考慮事項.... 2-7

## よ

- 読み取り専用ファイル・システム 1-4
- 弱いシンボル..... 6-28

## ら

- ライブラリ
  - ACL のルーチン ..... 7-8
  - TCB の一部として ..... 1-3
  - セキュリティ関連..... 1-5
  - ルーチン..... 1-7

## り

- リソース管理..... 8-33
- リターン値..... 8-49
- リファレンス..... 8-39
- リブレイ
  - 検出 ..... 8-20
  - 防止 ..... 8-32

## れ

- 例
  - ACL 許可の削除 ..... 7-14
  - ACL の継承..... 7-14
  - ACL の作成 ..... 7-10
  - audgenl() ..... 5-1

auditmask..... 5-6  
アプリケーション固有の監査レコー  
ド ..... 5-8  
監査 iovec タイプ・レコード .. 5-4  
監査タプルの解析マクロ ..... 5-16  
サイト定義監査イベント ..... 5-9

レルム..... C-3

## わ

---

ワン・タイム・パスワード .... C-36

## Tru64 UNIX ドキュメントの購入方法

Tru64 UNIX ドキュメントのご購入については、弊社担当営業または日本ヒューレット・パッカートの各営業所/代理店にお問い合わせください。

各ドキュメント・キットの注文番号は以下のとおりです。ドキュメント・キットに含まれるマニュアルの内容については『ドキュメント概要』を参照してください。

キット名	注文番号
Tru64 UNIX Documentation CD-ROM	QA-6ADAA-G8
Tru64 UNIX Documentation Kit	QA-6ADAA-GZ
End User Documentation Kit	QA-6ADAB-GZ
- Startup Documentation Kit	QA-6ADAC-GZ
- General User Documentation Kit	QA-6ADAD-GZ
- System and Network Management Documentation Kit	QA-6ADAE-GZ
Developer's Documentation Kit	QA-6ADAF-GZ
Reference Pages Documentation Kit	QA-6ADAG-GZ
TruCluster Server Documentation Kit	QA-6BRAA-GZ
Tru64 UNIX 日本語ドキュメント・キット	QA-6ADJB-GZ
スタートアップ・ドキュメント・キット	QA-6ADJC-GZ
一般ユーザ・ドキュメント・キット	QA-6ADJD-GZ
システム/ネットワーク管理ドキュメント・キット	QA-6ADJE-GZ
プログラミング・ドキュメント・キット	QA-6ADJF-GZ
CDE 翻訳ドキュメント・キット	QA-6ADJG-GZ
TruCluster Server 日本語ドキュメント・キット	QA-05SJA-GZ
Advanced Server for UNIX 日本語ドキュメント・キット	QA-5U2JA-GZ





# マニュアルに対するご意見

## Tru64 UNIX

セキュリティ・プログラミング・ガイド

AA-RTFKA-TE

弊社のマニュアルに関して、ご意見、ご要望、または内容の不明確な部分など、お気づきの点がございましたら、下記にご記入の上、弊社社員にお渡しくださるようお願い申し上げます。

マニュアルの採点：

	大変良い	良い	普通	良くない
正確さ(説明どおりに動作するか)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
情報量(十分か)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
分かり易さ	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
マニュアルの構成	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
図(役立つか)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
例(役立つか)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
索引(項目の検索性)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ページ・レイアウト(情報の検索性)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

内容の不明確な部分がありましたら、以下にご記入ください：

ペー ジ


その他お気づきの点がございましたら、以下にご記入ください：


ご使用のソフトウェアのバージョン： \_\_\_\_\_

貴社名/部課名 \_\_\_\_\_

御名前 \_\_\_\_\_

記入日 \_\_\_\_\_

(注) 当用紙を受け取った弊社社員は、すみやかに下記にお送りください。

ビジネスクリティカルシステム統括本部 **BCS** 技術本部 **Alpha** ソフトウェア技術部