Solaris Trusted Extensions 開発ガイド

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 819–7612–12
2008 年 4 月

原典: Solaris Trusted Extensions Developer's Guide

Part No: 819-0869-12

Revision A

080424@19860

目次

はじめに

Solaris Trusted Extensions 開発ガイドは、Solaris™ Trusted Extensions ソフトウェアが設定されたシステム用に信頼できる新しいアプリケーションを作成するための、アプリケーションプログラミングインタフェース (API) の使用法について説明します。読者は、UNIX® のプログラミングに慣れていて、セキュリティーポリシーの概念を理解している必要があります。

注 – Solaris のこのリリースでは、SPARC® および x86 系列のプロセッサアーキテクチャー (UltraSPARC®、SPARC64、AMD64、Pentium、および Xeon EM64T) を使用するシステムをサポートします。サポートされるシステムは、`http://www.sun.com/bigadmin/hcl` にある Solaris 10 ハードウェア互換リストに表示されます。本書では、プラットフォームにより実装が異なる場合は、それを特記します。

本書の x86 に関連する用語は、次のとおりです。

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

サポートされるシステムについては、Solaris 10 ハードウェア互換リストを参照してください。

このマニュアル内のプログラム例は、API の紹介を中心としていて、エラーチェックは行なっていません。実際に作成するアプリケーションでは、適切なエラーチェックを実行してください。

Solaris Trusted Extensions のマニュアルの構成

Solaris Trusted Extensions のマニュアルセットは、Solaris 10 5/08 リリースのマニュアルを補足します。Solaris Trusted Extensions を十分に理解するためには、両方のマニュアルのセットを参照してください。Solaris Trusted Extensions のマニュアルセットは、次のマニュアルで構成されています。

マニュアルのタイトル	トピック	対象読者
『Solaris Trusted Extensions 移行ガイド』	旧版。Trusted Solaris 8 ソフトウェア、Solaris 10 5/08 ソフトウェア、および Solaris Trusted Extensions ソフトウェアの違いについて概説しています。 このリリースでは、Trusted Extensions の変更点を『Solaris OS の概要』のマニュアルで概説しています。	すべてのユーザー
『Solaris Trusted Extensions リファレンスマニュアル』	旧版。Solaris 10 11/06 および Solaris 10 8/07 リリースの Trusted Extensions における Solaris Trusted Extensions マニュアルページが記載されています。 このリリースでは、Trusted Extensions のマニュアルページは Solaris のマニュアルページに含まれています。	すべてのユーザー
『Solaris Trusted Extensions ユーザーズガイド』	Solaris Trusted Extensions の基本的な機能を説明します。このマニュアルには用語集が含まれています。	一般ユーザー、管理者、開発者
『Solaris Trusted Extensions インストールと構成』	旧版。Solaris 10 11/06 と Solaris 10 8/07 リリースの Trusted Extensions における Solaris Trusted Extensions を計画、インストール、および構成する方法を説明しています。	管理者、開発者
『Solaris Trusted Extensions 構成ガイド』	Solaris 10 5/08 リリース以降において、Solaris Trusted Extensions を有効化、および最初に構成する方法を説明しています。旧版の『Solaris Trusted Extensions インストールと構成』に替わるものです。	管理者、開発者
『Solaris Trusted Extensions 管理の手順』	特定の管理タスクの実行方法を示します。	管理者、開発者
『Solaris Trusted Extensions 開発ガイド』	Solaris Trusted Extensions でアプリケーションを開発する方法を説明します。	開発者、管理者
『Solaris Trusted Extensions ラベルの管理』	ラベルエンコーディングファイルにラベル構成要素を指定する方法を説明します。	管理者
『コンパートメントモードワークステーションのラベル作成: エンコード形式』	ラベルエンコーディングファイルで使用する構文を説明します。 構文は、システムの正しい形式のラベルに関する規則を規定します。	管理者

内容の紹介

第 1 章では、Solaris Trusted Extensions API の概要を示し、セキュリティーポリシーがシステム内でどのように強制されるかを説明します。

第 2 章では、プロセスおよびデバイスオブジェクトに対してラベルを管理するためのデータ型および API について説明します。さらに、認可上限、プロセスが機密ラベルを取得する方法、ラベル操作で特権が必要な場合についても説明します。ラベルの処理に関するガイドラインも示します。

第 3 章では、ラベル用 API を使用するコーディング例を示します。

第 4 章では、ラベル API の使用例として Trusted Extensions のマルチレベル印刷サービスを使用します。

第 5 章では、同一のワークステーション内のプロセス間通信、およびネットワークにまたがるプロセス間通信に対してセキュリティーポリシーがどのように適用されるかの概要を示します。

第 6 章では、セキュリティー関連の X ウィンドウシステム情報に対して、管理アプリケーションがアクセスしたり変更したりするためのデータ型および API について説明します。この章にはコーディング例もあります。

第 7 章では、ラベルおよび認可上限を作成するためのグラフィカルユーザーインタフェース (GUI) を作成するためのデータ型および API について説明します。この章にはコーディング例もあります。

第 8 章では、インターネットを通じた攻撃から Web サーバーおよびその Web コンテンツを隔離する、安全な Web ブラウジングのプロトタイプの例を示します。

第 9 章では、Solaris Trusted Extensions ソフトウェアに備えられているラベル API をミラー化する Java™ クラスおよびメソッドの実験的設定について説明します。この章にはソースコードに対するポインタおよびビルド手順も記載されているので、これらの API を使用してラベル対応のアプリケーションを作成することができます。

付録 A では、Solaris Trusted Extensions マニュアルページ、共有ライブラリ、ヘッダーファイル、およびデータ型名とインタフェース名に使用される略号に関する情報を示します。アプリケーションのリリースの準備に関する情報も示します。

付録 B では、パラメータおよび戻り値の宣言を含む、プログラミングインタフェースのリストを示します。

マニュアル、サポート、およびトレーニング

Sun の Web サイトでは、次のサービスに関する情報も提供しています。

- マニュアル (http://jp.sun.com/documentation/)
- サポート (http://jp.sun.com/support/)
- トレーニング (http://jp.sun.com/training/)

表記上の規則

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

表 P–1 表記上の規則

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

コード例は次のように表示されます。

- C シェル

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

- C シェルのスーパーユーザー

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

- Bourne シェルおよび Korn シェル

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

- Bourne シェルおよび Korn シェルのスーパーユーザー

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

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

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

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

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

◆ ◆ ◆ 第 1 章

1 Solaris Trusted Extensions API とセキュリティーポリシー

Solaris™ Trusted Extensions ソフトウェア (Trusted Extensions) には、ラベルにアクセスしてその処理を行うアプリケーションの作成を可能にする、アプリケーションプログラミングインタフェース (API) が用意されています。この章では、その API の機能とTrusted Extensions セキュリティーポリシーの概要について説明します。

Trusted Extensions で使用する用語の定義については、『Solaris Trusted Extensions ユーザーズガイド』の用語集を参照してください。

Solaris オペレーティングシステム (Solaris OS) で Trusted Extensions API を使用する方法の例は、Solaris ソースコードを参照してください。OpenSolaris の Web サイト (`http://jp.opensolaris.org/`) の左のナビゲーションバーにある「Source Browser」をクリックします。Source Browser を使用して Solaris のソースコードを検索します。

この章の内容は次のとおりです。

ラベルについて

Solaris Trusted Extensions ソフトウェアは、Solaris OS のセキュリティー機能を拡張する一連のポリシーおよびサービスを提供します。この拡張機能によって、ラベルの関係に基づくアクセス制御が可能になります。

ラベルは、データへのアクセスを制御し、データの格付けを管理します。ラベルは、システムセキュリティーポリシーによって解釈される属性です。「システムセキュリティーポリシー」は、システムで処理される情報を保護するために、システムソフトウェアによって強制される一連の規則です。「セキュリティーポリシー」という用語は、ポリシーそのものを指す場合と、ポリシーの実装を指す場合があります。詳細は、24 ページの「Trusted Extensions セキュリティーポリシー」を参照してください。

この節では、ラベルの型、範囲、構成要素、および関係の概要を説明します。

ラベルの型

Trusted Extensions ソフトウェアは、機密ラベルと認可上限ラベルの 2 つの型のラベルを定義します。「機密ラベル」 は、エンティティーのセキュリティーレベルを示し、一般に「ラベル」と呼ばれます。「認可上限ラベル」は、ラベル範囲の上限を定義し、一般に「認可上限」と呼ばれます。

機密ラベル

Trusted Extensions ソフトウェアは、ゾーンを使用して、格付けされた情報をさまざまなレベルに収容します。各レベルは専用のゾーンに関連付けられ、ゾーンには機密ラベルが付けられます。機密ラベルは、ゾーンにおける情報の機密度を示し、ゾーン内のすべてのサブジェクトおよびオブジェクトに適用されます。ラベルには CONFIDENTIAL、SECRET、TOP SECRET などがあります。「サブジェクト」は、プロセスなどの能動的なエンティティーであり、 情報をオブジェクト間で移動させたり、システムの状態を変化させます。「オブジェクト」はファイルやデバイスなどの受動的なエンティティーであり、データを収容したり、データを受け取ったりします。ゾーンで実行されるすべてのプロセス、またゾーンに含まれるすべてのファイルなどは、そのゾーンと同じ機密ラベルを持っています。すべてのプロセスおよびオブジェクトは、必須アクセス制御 (MAC) の決定に使用される機密ラベルを持ちます。デフォルトでは、機密ラベルはウィンドウシステムで表示されます。

認可上限ラベル

セキュリティー管理者は、各ユーザーに認可上限を割り当てます。認可上限は、ラベル範囲の上限を定義するラベルです。たとえば、ユーザーの認可上限が SECRET である場合、そのレベル以下に格付けされる情報にはアクセスできますが、それより上のレベルに格付けされる情報にはアクセスできません。「ユーザー認可上限」は、セキュリティー管理者によって割り当てられます。これは、ユーザーがセッション中にファイルにアクセスでき、プロセスを開始できる、最高位のラベルです。すなわち、ユーザー認可上限は、ユーザーのアカウントラベル範囲の上限です。ログイン時に、ユーザーはセッション認可上限を選択します。「セッション認可上限」によって、ユーザーがアクセスできるラベルが決まります。セッション認可上限は、ユーザーがそのログインセッション中にファイルにアクセスでき、プロセスを開始できる「最小上限」を定めます。セッション認可上限は、ユーザー認可上限によって決定されます。

ラベル範囲

セキュリティー管理者は、「必須アクセス制御」(MAC) ポリシを強制するためにラベル範囲およびラベルセットを定義します。「ラベル範囲」は、認可上限によって上端が区切られ、最小ラベルによって下端が区切られるラベルのセットです。「ラ

ベルリミット」は、ラベル範囲の上限です。「ラベルセット」には、互いに共通要素がない場合もある 1 つ以上の別個のラベルが含まれます。ラベルセットの中のラベルには、どちらが優位であるかという関係はありません。

ラベルの構成要素

ラベルには、階層的な格付け、およびゼロ個以上の階層的でないコンパートメントのセットが含まれます。格付けは「レベル」または機密レベルとも呼ばれます。「格付け」は、TOP SECRET、UNCLASSIFIED など、ラベルの階層内の 1 つのレベルを表します。「コンパートメント」は、格付けと関連付けられ、人事 (HR) 部や営業部の個人情報など、システム内の別個の階層的でない領域の情報を表します。コンパートメントは、特定領域の情報を知る必要があるユーザーにのみアクセスを制限します。たとえば、SECRET 格付けのユーザーのみが、コンパートメントの関連リストによって指定される機密情報にアクセスできますが、その他の機密情報にはアクセスできません。格付けとコンパートメントはともに、ゾーンのラベルとゾーン内のリソースを表します。

label_encodings ファイルに指定されるテキスト形式の格付けは、次のとおりです。

```
CLASSIFICATIONS:
name= CONFIDENTIAL; sname= C; value= 4; initial compartments= 4-5 190-239;
name= REGISTERED; sname= REG; value= 6; initial compartments= 4-5 190-239;
```

label_encodings ファイルに指定されるテキスト形式のコンパートメントは、次のとおりです。

```
WORDS:
name= HR; minclass= C; compartments= 0;
```

ラベルの定義および形式についての詳細は、『Solaris Trusted Extensions ラベルの管理』および『コンパートメントモードワークステーションのラベル作成: エンコード形式』を参照してください。ラベル API については、第 2 章を参照してください。

ラベル関係

ラベルを比較することは、プロセスのラベルをターゲットのラベルと比較することであり、ターゲットは機密ラベルの場合も認可上限ラベルの場合もあります。比較の結果に基づいて、プロセスはオブジェクトへのアクセスを許可または拒否されます。プロセスのラベルがターゲットのラベルより優位である場合にのみ、アクセスが許可されます。ラベル関係と優位性については、この節で説明します。例は、50 ページの「2 つのラベル間の関係の判別」を参照してください。

「機密レベル」は数値による格付けです。ラベルは、エンティティーの機密レベルを示し、ゼロ個以上のコンパートメントを含むことがあります。エンティティーとは、プロセス、ゾーン、ファイル、デバイスなどのことであり、ラベルを付けることができます。

ラベルには次のような型があって、相互に一定の関係があります。

- 等位 – あるラベルがほかのラベルと等しい場合、次のいずれも当てはまります。
  - 格付けの数値がほかのラベルの格付けと等しい。
  - ほかのラベルとまったく同じコンパートメントを持つ。
- 優位 – あるラベルがほかのラベルより優位である場合、次のいずれも当てはまります。
  - 格付けの数値がほかのラベルの格付け以上。
  - ほかのラベルとまったく同じコンパートメントを持つ。
- 厳密優位 – あるラベルがほかのラベルより厳密に優位である場合、次のいずれも当てはまります。
  - 格付けの数値がほかのラベルの格付け以上。
  - ほかのラベルにあるすべてのコンパートメントを持ち、さらに 1 つ以上のその他のコンパートメントを持つ。
- 分離 – あるラベルがほかのラベルから分離している場合、次のいずれも当てはまります。
  - ラベルが等位でない。
  - ラベルがほかのラベルより優位にない。

ラベルの格付けおよびコンパートメントの指定には、`label_encodings` ファイルが使用されます。`label_encodings(4)` のマニュアルページを参照してください。

いずれかの型のラベルに別のラベル以上の機密レベルがある場合、そのラベルは別のラベルより優位であると言います。セキュリティーレベルに関するこの比較は、ラベルの格付けとコンパートメントに基づきます。優位ラベルの格付けは、別のラベルの格付け以上にする必要があります。さらに、優位ラベルには別のラベルのすべてのコンパートメントが含まれている必要があります。2 つのラベルが同等な場合は、「互いに優位である」と言います。

`label_encodings` ファイルの次の抜粋例では、`REGISTERED (REG)` ラベルは `CONFIDENTIAL (C)` ラベルより優位です。比較は各ラベルの `value` キーワードの値に基づきます。`REG` ラベルの `value` キーワードの値は数値的に `C` ラベルの `value` キーワードの値以上です。両方のラベルは `PUBLIC (P)` ラベルより優位です。

`initial compartments` キーワードの値は、格付けに最初に関連付けられるコンパートメントのリストを示します。`initial compartments` キーワードの各数字は「コンパートメントビット」であり、それぞれが特定のコンパートメントを表します。

```
CLASSIFICATIONS:
name= PUBLIC; sname= P; value= 1;
name= CONFIDENTIAL; sname= C; value= 4; initial compartments= 4-5 190-239;
name= REGISTERED; sname= REG; value= 6; initial compartments= 4-5 190-239;
```

次の label_encodings の抜粋では、REG HR ラベル (人事) が REG ラベルより優位です。REG HR ラベルには、REGISTERED 格付けと HR コンパートメントがあります。HR コンパートメントの compartments キーワードで 0 コンパートメントビットが設定されるので、REG HR 格付けはコンパートメント 0、 4–5、および 190–239 セットとなり、REG 格付けによって設定されるコンパートメントより値が大きくなります。

```
CLASSIFICATIONS:
name= REGISTERED; sname= REG; value= 6; initial compartments= 4-5 190-239;
...
WORDS:
name= HR; minclass= C; compartments= 0;
```

オブジェクトにアクセスするために厳密優位が必要な場合があります。前の例で、REG ラベルは P ラベルより厳密に優位であり、REG HR ラベルは REG ラベルより厳密に優位です。ラベルを比較すると、REG ラベルは別の REG ラベルより優位です。

互いに優位でないラベルは、分離していると言います。「分離」ラベルは、会社内の部門を分けるために使用されることがあります。次の例では、REG HR ラベル (人事) は REG Sales ラベルから分離していると定義されます。これらのラベルは、それぞれのコンパートメントによって異なるコンパートメントビットが設定されるために、分離されます。

```
CLASSIFICATIONS:
name= REGISTERED; sname= REG; value= 6; initial compartments= 4-5 190-239;
...
WORDS:
name= HR; minclass= C; compartments= 0;
name= Sales; minclass= C; compartments= 1;
```

ラベル API については、22 ページの「機密ラベル API」を参照してください。

Trusted Extensions API

この節では、このマニュアルで説明される次の 3 つの Trusted Extensions API の概要を説明します。

- ラベル API
- トラステッド X ウィンドウシステム API
- ラベルビルダー API

これらの Trusted Extensions API のほかに、Solaris OS で使用できるセキュリティー API を使用できます。Trusted Extensions で実行されるアプリケーションでは、その他

のセキュリティー属性の操作が必要な場合があります。たとえば、ユーザーおよびプロファイルデータベースには、ユーザー、役割、承認、およびプロファイルに関する情報が格納されています。これらのデータベースは、だれがプログラムを実行できるかを制限できます。さまざまな Solaris プログラム、さらに他社製アプリケーションに特権をコーディングすることができます。

これらの Solaris OS セキュリティー API に関する詳細は、『Solaris セキュリティーサービス開発ガイド』の第 2 章「特権付きアプリケーションの開発」を参照してください。

Solaris OS は、データの所有者がデータへのアクセスをだれに許可するかを決定する「任意アクセス制御」(DAC) を提供します。Trusted Extensions ソフトウェアは、必須アクセス制御 (MAC) と呼ばれる追加のアクセス制御を提供します。MAC では、一般ユーザーは「セキュリティーポリシー」を指定したり上書きしたりできません。セキュリティー管理者がセキュリティーポリシーを設定します。

アプリケーションは Trusted Extensions API を使用して、ホスト、ゾーン、および役割のラベルを取得します。セキュリティーポリシーが許可する場合、API によってユーザープロセスまたは役割プロセスに対してラベルを設定できます。ゾーンまたはホストに対するラベルの設定は、管理上の手続きであり、プログラム化する手続きではありません。

ウィンドウラベルをカスタマイズするためにアプリケーションを作成できます。Trusted Extensions ソフトウェアは、ラベル作成の基本的なユーザーインタフェースをアプリケーションに追加するための Motif ベースのプログラミングインタフェースを提供します。ラベル作成インタフェースによって、ユーザーは有効な機密ラベルおよび認可上限を対話形式で作成できます。

ラベル API は不透明なラベルで機能します。「不透明なラベル」では、ラベルの内部構造は外に現れません。不透明なラベルを使用することによって、ラベルの内部構造が変化しても、API で作成した既存のプログラムが機能するようにできます。たとえば、ラベルの特定ビットを配置するためにラベル API を使用することはできません。ラベル API では、ラベルの取得およびラベルの設定が可能です。セキュリティーポリシーによって許可されている場合のみ、ラベルの設定を行えます。

ラベル API

ラベル、ラベル範囲、およびラベル制限は、Trusted Extensions が設定されているシステム上の情報にだれがアクセスできるかを決定します。

ラベル API を使用して、ラベル、ラベル範囲と制限、およびラベル関係に対してアクセス、変換、および比較を実行します。ラベルは別のラベルより優位になったり、別のラベルから分離したりできます。

label_encodings ファイルには、それぞれの Trusted Extensions 環境に適した機密ラベル、認可上限ラベル、ラベル範囲、およびラベル関係が定義されています。このファイルはラベルの表示も制御します。セキュリティー管理者が label_encodings ファイルの作成および管理を担当します。label_encodings(4) のマニュアルページを参照してください。

プロセスのラベルは、プロセスが実行されるゾーンによって決定されます。

すべてのオブジェクトはラベルに関連付けられ、ラベル範囲に関連付けられる場合もあります。オブジェクトは、定義されたラベル範囲内の特定ラベルでアクセスできます。ラベル範囲に関連付けられるオブジェクトは、次のとおりです。

- すべてのユーザーとすべての役割
- 通信が許可されているすべてのホスト
- ゾーンインタフェースとネットワークインタフェース
- 割り当て可能なデバイス (テープドライブ、フロッピーディスクドライブ、CD-ROM デバイス、オーディオデバイスなど)
- 割り当てできないその他のデバイス (プリンタ、ワークステーションなど)

  ワークステーションアクセスは、フレームバッファーまたはビデオディスプレイデバイスに対して設定されるラベル範囲によって制御されます。セキュリティー管理者はこの範囲をデバイスマネージャー GUI を使用して設定します。デフォルトでは、デバイスの範囲は ADMIN_LOW から ADMIN_HIGH までです。

ラベルについての詳細は、16 ページの「ラベルの型」を参照してください。

アクセス制御の決定にラベルを使用する

MAC は、ラベルを持つアプリケーションを実行するプロセスのラベル、またはプロセスがアクセスしようとするオブジェクトのラベル範囲を比較します。MAC は、プロセスが下位ラベルを読み取るのを許可し、等位ラベルに書き込むのを許可します。

```
Label[Process] >= Label[Object]
```

マルチレベルポート (MLP) に対するプロセス制約は、複数ラベルの要求を待機し、要求の発信元に応答を送信できます。Trusted Extensions では、そのような応答は等位書き込みです。

```
Label[Process] = Label[Object]
```

ラベル API の型

機密ラベル API

機密ラベル API は、次のために使用できます。

- プロセスラベルの取得
- ラベルの初期化
- 2 つのラベルにおける最大の下限または最小の上限の検索
- 優位と等位のラベルの比較
- ラベルの型の検査と設定
- 可読形式へのラベルの変換
- `label_encodings` ファイルからの情報の取得
- 機密ラベルが有効であり、システム範囲内にあることの検査

この API については、第 2 章を参照してください。

認可上限ラベル API

ユーザー、デバイス、およびネットワークインタフェースにはラベル範囲があります。範囲の上限は、実質上の認可上限です。範囲の上限と範囲の下限が同じである場合、範囲は単一のラベルです。

認可上限ラベル API は、次のために使用できます。

- 2 つのラベルにおける最大の下限または最小の上限の検索
- 優位と等位のラベルの比較
- 内部形式と 16 進形式との認可上限の変換

この API については、第 2 章を参照してください。

ラベル範囲 API

ラベル範囲は、次に対して制限を設定するために使用します。

- ホストが情報を送受信するラベル
- ユーザーおよび役割に代わって動作するプロセスがシステム上で機能するラベル
- ユーザーがデバイスを割り当てるラベル

  ラベル範囲を使用することによって、ファイルがデバイスのストレージメディアに書き込まれるラベルが制限されます。

ラベル範囲は管理のために割り当てられます。ラベル範囲は、ユーザー、役割、ホスト、ゾーン、ネットワークインタフェース、プリンタ、およびその他のオブジェクトに適用できます。

次の方法によって、ラベル範囲に関する情報を取得できます。

- `getuserrange()` はユーザーのラベル範囲を取得します。
- `getdevicerange()` はデバイスのラベル範囲を取得します。
- `tninfo -t` *template-name* は、ネットワークインタフェースに関連付けられているテンプレートのラベル範囲を表示します。

この API については、第 2 章を参照してください。

トラステッド X ウィンドウシステム API

トラステッド X ウィンドウシステム, Version 11 サーバーがログイン時に起動します。このサーバーは、信頼できるプロセス間通信 (IPC) パスを使用してワークステーションウィンドウシステムを処理します。ウィンドウ、プロパティー、セレクション、および ToolTalk™ のセッションが複数の機密ラベルで別個のオブジェクトとして作成されます。この複数の機密ラベルでの別個のオブジェクトの作成は、「多インスタンス化」と呼ばれます。Motif ウィジェット、Xt イントリンシクス、Xlib、およびデスクトップインタフェースによって作成されるアプリケーションは、セキュリティーポリシーの制約内で実行されます。この制約は X11 プロトコルに対する拡張によって強制されます。

第 6 章では、24 ページの「Trusted Extensions セキュリティーポリシー」に示されるセキュリティー属性情報にアクセスできる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。このプログラミングインタフェースは、ラベルおよび認可上限をテキストに変換する場合にも使用できます。そのテキストは、トラステッド X ウィンドウシステムで表示するために指定した幅およびフォントリストによって制限できます。

トラステッド X ウィンドウシステムは、次のセキュリティー属性を格納します。

監査 ID	トラステッドパスフラグ
グループ ID	トラステッドパスウィンドウ
インターネットアドレス	ユーザー ID
プロセス ID	X ウィンドウサーバー所有者 ID
機密ラベル	X ウィンドウサーバー認可上限
セッション ID	X ウィンドウサーバー最小ラベル

トラステッドパスフラグは、トラステッドパスウィンドウとしてウィンドウを指定します。トラステッドパスウィンドウは、信頼できないプログラムによってシステムがアクセスされないように保護します。このウィンドウは、スクリーンストライプ、ログインウィンドウなどのように、常に最上位に位置するウィンドウです。

付録 B に、X11 トラステッド IPC パスを作成するために使用できる拡張機能がリストされています。

ラベルビルダー API

Trusted Extensions ソフトウェアは、アプリケーションのためにグラフィカルユーザーインタフェース (GUI) を作成できるラベルビルダー API を提供します。GUI はユーザー入力を受け取って、その入力から有効なラベルを作成します。

Solaris Trusted Extensions が構成されているシステム は、ラベル作成の基本的なユーザーインタフェースをアプリケーションに追加するための Motif ベースのプログラミングインタフェースを提供します。ラベル作成インタフェースによって、ユーザーは有効な機密ラベルおよび認可上限を対話形式で作成できます。このプログラミングインタフェースについては、第 7 章を参照してください。

Trusted Extensions セキュリティーポリシー

機密ラベルはデータへのアクセスを制御し、データの格付けを管理します。すべてのプロセスおよびオブジェクトには、MAC の決定に使用される機密ラベルがあります。ラベルは、システムセキュリティーポリシーによって解釈される属性です。「システムセキュリティーポリシー」は、システムで処理される情報を保護するためにシステムソフトウェアによって強制される一連の規則です。

このあとの節で、Trusted Extensions セキュリティーポリシーがマルチレベルの操作、ゾーン、およびラベルに対してどのように影響を及ぼすかについて説明します。

マルチレベル操作

複数の機密レベルで実行される操作を作成する場合、次の問題を考慮する必要があります。

- 大域ゾーンにおける下位書き込みポリシー
- デフォルトのセキュリティー属性
- デフォルトのネットワークポリシー
- マルチレベルポート
- MAC 適用外ソケット

大域ゾーンのプロセスのみが指定ラベルでプロセスを開始できるので、複数の機密レベルで実行される操作は大域ゾーンによって制御されます。

大域ゾーンにおける下位書き込みポリシー

下位のラベルのオブジェクトを書き込むプロセスなどのサブジェクトの機能は「下位書き込み」と呼ばれます。大域ゾーンにおける下位書き込みポリシーは管理のために指定されます。大域ゾーンプロセスは `ADMIN_HIGH` ラベルで実行されるので、ほ

かのラベルに関連付けられている特定のファイルシステムは、大域ゾーンで読み取り/書き込みマウントが可能です。ただし、このような特別なファイルシステムマウントは、管理のための自動マウントマップで指定する必要があり、大域ゾーンオートマウンタによってマウントされなければなりません。このマウントには、エクスポートされたファイルシステムと同じラベルを持つゾーンのゾーンパス内にマウントポイントがなければなりません。そのマウントポイントは、ラベル付けされたゾーン内から可視ではあってはいけません。

たとえば、`PUBLIC` ゾーンに `/zone/public` のゾーンパスがある場合、`/zone/public/home/mydir` の書き込み可能なマウントポイントが許可されます。それに対し、`/zone/public/root/home/mydir` の書き込み可能なマウントポイントは、ラベル付けされたゾーンによってアクセスできるが、大域ゾーンによってはアクセスできないので許可されません。クロスゾーン NFS マウントは許可されません。すなわち、NFS マウントのファイルは、そのファイルシステムをマウントしたゾーンで実行されるプロセスによってのみアクセスできます。大域ゾーンプロセスは、標準の任意アクセス制御 (DAC) ポリシーに従って、そのようなファイルに下位書き込みできます。

ゾーンに関連付けられたローカルファイルシステムは、ファイル「アクセス権」とアクセス制御リスト (ACL) を使用する DAC によって、大域ゾーンプロセスによるアクセスから保護されます。各ゾーンのルート (`/`) ディレクトリの親ディレクトリは、`root` プロセスまたは `file_dac_search` 特権を表明するプロセスのみによってアクセス可能です。

一般に、大域ゾーンから下位書き込みできるかどうかは制限されています。下位書き込みが使用されるのは、通常、`setflabel()` インタフェースを使用してファイルが再格付けされる場合、または特権ユーザーが異なるゾーンでファイルマネージャーアプリケーション間をファイルをドラッグ＆ドロップする場合のみです。

デフォルトのセキュリティー属性

デフォルトのセキュリティー属性は、ほかの「ホストタイプ」から Trusted Extensions ホストに届くメッセージに割り当てられます。この属性の割り当ては、ネットワークデータベースファイルの設定に従って行われます。ホストタイプ、サポートされるセキュリティー属性、およびネットワークデータベースファイルのデフォルト設定については、『Solaris Trusted Extensions 管理の手順』を参照してください。

デフォルトのネットワークポリシー

データを送受信するネットワーク操作の場合、デフォルトポリシーとして、ローカルプロセスと遠隔ピアが同じラベルを持っている必要があります。このポリシーは、ネットワークラベルが `ADMIN_LOW` である、大域ゾーンを含むすべてのゾーンに適用されます。ただし、このデフォルトのネットワークポリシーは、ファイルシステムをマウントするためのポリシーより柔軟性があります。Trusted Extensions には、デフォルトのネットワークポリシーを上書きするための管理インタフェースおよび

プログラマチックインタフェースがあります。たとえば、システム管理者は、大域ゾーンまたはラベル付けされたゾーンに MLP を作成して、異なるラベルでの待機を可能にすることができます。

マルチレベルポート

注意 – マルチレベルポートを使用して MAC ポリシーに違反するには、十分な注意が必要です。このメカニズムを使用する場合、サーバーアプリケーションで MAC ポリシーが強制されていることを確認してください。

マルチレベルポート (MLP) は、`tnzonecfg` 管理データベースにリストされています。ゾーン内のプロセスは、`net_bindmlp` 特権を表明する場合、MLP にバインドできます。ポート番号が 1024 より小さい場合、`net_privaddr` 特権の表明が必要です。このバインドによって、プロセスは、バインドされる IP アドレスに関連付けられているすべてのラベルで接続を受け入れることができます。ネットワークインタフェースに関連付けるラベルは、`tnrhdb` データベースおよび `tnrhtp` データベースで指定します。ラベルの指定は、範囲、明示的列挙のラベル、または両方の組み合わせによって行えます。

MLP にバインドされている特権プロセスが TCP 要求を受信すると、要求元のラベルで応答が自動的に送信されます。UDP データグラムの場合、`SO_RECVUCRED` オプションによって指定されるラベルで応答が送信されます。

特権プロセスは、要求のラベルとほかのパラメータを比較することによって、より制限の多い MAC ポリシーを実装できます。たとえば、Web サーバーは、要求プロセスのラベルと URL で指定されるファイルのラベルを比較できます。`getpeerucred()` 関数を使用すると、遠隔ピアの資格が返されるので、遠隔ラベルを決定できます。同じホストのゾーンでピアが実行されている場合、`ucred_get()` ライブラリルーチンによって資格のフルセットが返されます。ピアがローカルであるか遠隔であるかにかかわらず、`ucred_getlabel()` 関数を使用することによって、`ucred` データ構造からピアのラベルにアクセスできます。このラベルは、`bldominates()` などの関数によってほかのラベルと比較できます。

ゾーンは単一レベルポートおよびマルチレベルポートにすることができます。65 ページの「マルチレベルポートについて」を参照してください。

MAC 適用外ソケット

Trusted Extensions ソフトウェアには、下位ラベルの終端と通信するためにソケットを使用できるように指定する明示的なソケットオプション `SO_MAC_EXEMPT` があります。

注意 – SO_MAC_EXEMPT ソケットオプションは絶対に誤って使用しないでください。このソケットオプションを使用して MAC ポリシーを無効にする際には、十分に注意してください。このメカニズムを使用する場合、クライアントアプリケーションで MAC ポリシーが強制されていることを確認してください。

Trusted Extensions ソフトウェアは、SO_MAC_EXEMPT オプションの使用を次のように制限します。

- このソケットオプションを明示的に設定するには、プロセスは net_mac_aware 特権を表明する必要があります。
- このソケットオプションの使用をさらに制限するために、一般ユーザーの制限セットから net_mac_aware 特権を削除できます。

詳細は、user_attr(4) のマニュアルページを参照してください。

ソケットがライブラリによって管理される場合など、このソケットオプションを明示的に設定するのは実用的でないことがあります。このような状況では、ソケットオプションを暗黙に設定できます。setpflags() システムコールによって、NET_MAC_EXEMPT プロセスフラグを設定できます。このプロセスフラグを設定するには、net_mac_exempt 特権も必要です。プロセスフラグが有効な間に開かれているすべてのソケットは、SO_MAC_EXEMPT ソケットオプションが自動的に設定されます。setpflags(2) および getpflags(2) のマニュアルページを参照してください。

変更または再コンパイルできないアプリケーションの場合、ppriv -M コマンドを使用して net_mac_exempt プロセスフラグをアプリケーションに渡します。この場合、そのアプリケーションによって開かれるすべてのソケットは SO_MAC_EXEMPT オプションが設定されます。ただし、アプリケーションの子プロセスには、このプロセスフラグや関連特権はありません。

SO_MAC_EXEMPT ソケットオプションを使用する必要がある場合、可能であれば必ずアプリケーションのソースコードを精査して修正してください。コードをそのように変更できない場合、またより安全な方法がない場合は、ppriv -M コマンドを使用することもできます。

SO_MAC_EXEMPT ソケットオプションは Solaris OS では多用されていません。このオプションは NFS クライアントによって使用されていました。NFS クライアントは、信頼できないオペレーティングシステムで異なるラベルで実行される NFS サーバーと通信する必要のある場合があります。NFS クライアントは MAC ポリシーを強制することによって、不適切な要求が認められないようにします。

Solaris OS では、NFS のサーバーコードとクライアントコードに MAC ポリシーが組み込まれて強制されるので、Solaris クライアントまたはサーバーと信頼できないクライアントまたはサーバーとの通信で MAC ポリシーが有効になります。信頼できないホストが Trusted Extensions を実行するシステムと通信できるようにするには、信頼

できないホストのエントリが tnrhdb データベースにある必要があります。詳細は、『Solaris Trusted Extensions 管理の手順』の「トラステッドネットワークデータベースの構成 (作業マップ)」を参照してください。

注 – Trusted Extensions API を Solaris OS で使用する方法の例は、Solaris ソースコードを参照してください。OpenSolaris の Web サイト (http://jp.opensolaris.org/) の左のナビゲーションバーにある「Source Browser」をクリックします。Source Browser を使用して Solaris のソースコードを検索します。

ゾーンとラベル

Trusted Extensions が設定されているシステムのすべてのオブジェクトは、ゾーンに関連付けられています。そのゾーンは「ラベル付けされたゾーン」と呼ばれます。ラベル付けされたゾーンは非大域ゾーンであり、一般ユーザーがアクセスできます。複数のラベルでクリアされるユーザーは、その各ラベルでゾーンにアクセスすることができます。

「大域ゾーン」は、システムのセキュリティーポリシーを制御するファイルおよびプロセスを含む特別なゾーンです。大域ゾーンのファイルには、役割および特権プロセスのみがアクセスできます。

大域ゾーンのラベル

大域ゾーンはラベルの範囲に割り当てられます。その範囲は ADMIN_LOW から ADMIN_HIGH までです。ADMIN_HIGH と ADMIN_LOW は「管理ラベル」です。

ほかのゾーンと共有される大域ゾーンのオブジェクトには ADMIN_LOW ラベルが割り当てられます。たとえば、/usr、/sbin、および/lib ディレクトリのファイルには ADMIN_LOW ラベルが割り当てられます。これらのディレクトリとその内容はすべてのゾーンで共有されます。これらのファイルおよびディレクトリは、通常、パッケージからインストールされ、パッケージまたはパッチの手順の場合を除いて一般に変更されません。ADMIN_LOW ファイルを変更するには、通常、プロセスがスーパーユーザーによって、またはすべての特権を持つユーザーによって実行される必要があります。

大域ゾーン専用の情報には ADMIN_HIGH ラベルが割り当てられます。たとえば、大域ゾーンのすべてのプロセス、および /etc ディレクトリのすべての管理ファイルには ADMIN_HIGH ラベルが割り当てられます。役割に関連付けられているホームディレクトリには ADMIN_HIGH ラベルが割り当てられます。ユーザーに関連付けられているマルチレベル情報にも ADMIN_HIGH ラベルが割り当てられます。24 ページの「マルチレベル操作」を参照してください。大域ゾーンへのアクセスは制限されています。システムサービスおよび管理役割のみが大域ゾーンのプロセスを実行できます。

ラベル付けされたゾーン

非大域ゾーンは「ラベル付けされたゾーン」と呼ばれます。ラベル付けされたゾーンにはそれぞれ固有のラベルがあります。ラベル付けされたゾーン内のすべてのオブジェクトのラベルは同じです。たとえば、ラベル付けされたゾーンで実行されるすべてのプロセスのラベルは同じです。ラベル付けされたゾーンの書き込み可能なすべてのファイルのラベルも同じです。複数のラベルでクリアされるユーザーは、ラベル付けされたゾーンに各ラベルでアクセスできます。

Trusted Extensions は、ゾーン用のラベル API のセットを定義します。次の API は、ラベル付けされたゾーンに関連付けられているラベル、およびゾーン内のパス名を取得します。

- `getpathbylabel()`
- `getzoneidbylabel()`
- `getzonelabelbyid()`
- `getzonelabelbyname()`
- `getzonerootbyid()`
- `getzonerootbylabel()`
- `getzonerootbyname()`

これらの API の詳細は、38 ページの「ゾーンのラベルへのアクセス」を参照してください。

ファイルのラベルは、ゾーンのラベル、またはファイルを所有するホストのラベルに基づきます。したがって、ファイルの再ラベル付けを行う場合、該当するラベル付けされたゾーンまたはラベル付けされたホストにファイルを移動する必要があります。ファイルの再ラベル付けのプロセスは、ファイルの「再格付け」とも呼ばれます。`setflabel()` ライブラリルーチンは、ファイルを移動することによってファイルの再ラベル付けを行います。ファイルの再ラベル付けのためには、プロセスが `file_upgrade_sl` 特権または `file_downgrade_sl` 特権を表明する必要があります。`getlabel(2)` および `setflabel(3TSOL)` のマニュアルページを参照してください。

特権の設定に関する詳細は、『Solaris セキュリティーサービス開発ガイド』の第 2 章「特権付きアプリケーションの開発」を参照してください。

◆ ◆ ◆ 第 2 章

2 ラベルと認可上限

この章では、ラベルの初期化、ラベルと認可上限との比較など、基本的なラベル操作を実行するための Solaris Trusted Extensions API について説明します。また、プロセスのラベルにアクセスするための API についても説明します。

Trusted Extensions API を Solaris OS で使用する方法の例は、Solaris ソースコードを参照してください。OpenSolaris の Web サイト (`http://jp.opensolaris.org/`) の左のナビゲーションバーにある「Source Browser」をクリックします。Source Browser を使用して Solaris のソースコードを検索します。

この章の内容は次のとおりです。

第 3 章に、この章で説明するプログラミングインタフェースのコーディング例があります。

特権操作とラベル

操作がセキュリティーポリシーをバイパス、つまり上書きする場合、操作はその実効セットに特別な特権が必要です。

特権は、プログラム上または管理上、次のように実効セットに追加されます。

- 実行可能ファイルが root によって所有され、セットユーザー ID のアクセス権ビットが設定されている場合、実効セットにすべての特権を含んで起動されます。たとえば、CDE ファイルマネージャーが実効セットにすべての特権を含んで起動されます。次に、ファイルマネージャーはそのほとんどの特権をプログラム上で放棄し、ラベル間のドラッグ＆ドロップ操作の実行に必要な特権のみを保持します。
- 管理者が、SMF サービス用のマニフェストファイル、または一般コマンド用の RBAC データベース exec_attr ファイルに特権を指定できます。このファイルについての詳細は、exec_attr(4) のマニュアルページを参照してください。

バイナリラベルを変換する場合、および機密ラベルをアップグレードまたはダウングレードする場合、操作には特別な特権が必要です。

ユーザーおよび役割は、特別な特権によって操作を実行できます。その特権は「権利プロファイル」を使用することによって指定できます。特定の特権によって特定の関数を実行できるようにアプリケーションを作成することもできます。特別な特権を必要とするアプリケーションを作成する場合、特権を必要とする関数を実行するときのみ特権を有効にし、関数が完了したら特権を削除するようにします。この方法を「特権ブラケット」と呼びます。詳細は、『Solaris セキュリティーサービス開発ガイド』を参照してください。

- バイナリラベルの変換 – ラベルの内部表現と文字列を相互に変換できます。プロセスのラベルが変換されるラベルより優位ではない場合、変換を実行するために、呼び出し元プロセスには sys_trans_label 特権が必要です。
- 機密ラベルのアップグレードまたはダウングレード – ファイルの機密ラベルを「ダウングレード」または「アップグレード」できます。ファイルが呼び出し元プロセスに所有されていない場合、呼び出し元プロセスには file_owner 特権がその実効セットに必要です。詳細は、setflabel(3TSOL) のマニュアルページを参照してください。

  プロセスは、その実効セットに file_downgrade_sl 特権を含めることによって、ファイルシステムオブジェクトの既存の機密ラベルをそれより優位ではない新しい機密ラベルに設定できます。 file_downgrade_sl 特権によって、ファイルを分離ラベルに付け替えることができます。

  プロセスは、その実効セットに file_upgrade_sl 特権を含めることによって、ファイルシステムオブジェクトの既存の機密ラベルをそれより優位である新しい機密ラベルに設定できます。

アプリケーションは次のいずれかで動作するので、ほとんどのアプリケーションは特権を使用してアクセス制御をバイパスしません。

- アプリケーションは 1 つの機密ラベルで起動され、その同じ機密ラベルでオブジェクトのデータにアクセスします。
- アプリケーションは 1 つの機密ラベルで起動され、ほかの機密ラベルでオブジェクトのデータにアクセスしますが、必須アクセス操作がシステムセキュリティーポリシーによって許可されます。たとえば、下位読み取りが MAC によって許可されます。

アプリケーションがそのプロセスの機密ラベルとは異なる機密ラベルでデータにアクセスしようとして拒否される場合、プロセスにはアクセスを得るための特権が必要です。「特権」によって、アプリケーションは MAC または DAC をバイパスできます。たとえば、`file_dac_read`、`file_dac_write`、および `file_dac_search` 特権は DAC をバイパスします。`file_upgrade_sl` および `file_downgrade_sl` 特権は MAC をバイパスします。どのようにアクセスするにしても、アクセスされるデータの格付けが、アプリケーションの設計によって損なわれてはなりません。

アプリケーションがそれ自体の機密ラベルまたは別のオブジェクトの機密ラベルを変更する場合、必ずすべてのファイル記述子を閉じてください。開いているファイル記述子があると、機密データがほかのプロセスに漏れる可能性があります。

ラベル API

この節では、基本的なラベル操作に使用できる API について説明します。それらの API を使用するには、次のヘッダーファイルを組み込む必要があります。

```
#include <tsol/label.h>
```

ラベル API は `-ltsol` ライブラリオプションを指定してコンパイルします。

Trusted Extensions API は、次に対するデータ型を含みます。

- 機密ラベル – `m_label_t` 型定義は機密ラベルを表します。`m_label_t` 構造体は不透明です。

  インタフェースは `m_label_t` 型の変数をパラメータとして受け入れます。インタフェースは `m_label_t` 型の変数で機密ラベルを返します。`m_label_t` 型定義は `blevel_t` 構造体と互換性があります。
- 機密ラベル範囲 – `brange_t` データ構造は機密ラベルの範囲を表します。この構造体は最小ラベルと最大ラベルを保持します。構造体フィールドは `variable.lower_bound` と `variable.upper_bound` です。

次の操作のための API について、この節で説明します。

- Trusted Extensions システムの検出
- プロセス機密ラベルへのアクセス
- ラベルのためのメモリーの割り当てと解放
- ファイルのラベルの取得と設定
- ラベル範囲の取得
- ゾーンのラベルへのアクセス
- 遠隔ホストタイプの取得
- ラベルと文字列との変換
- ラベルの比較

Trusted Extensions システムの検出

`is_system_labeled()` ルーチンを使用して、Trusted Extensions システム上で実行中であるかどうかを判別します。次のルーチンの記述には、各ルーチンのプロトタイプ宣言が含まれます。

```
int is_system_labeled(void);
```

Trusted Extensions ソフトウェアがインストールされてアクティブである場合、`is_system_labeled()` ルーチンは `TRUE` (`1`) を返します。それ以外の場合は `FALSE` (`0`) を返します。

`is_system_labeled(3C)` のマニュアルページを参照してください。このルーチンの使用例は、57 ページの「`get_peer_label()` ラベル対応関数」を参照してください。

ほかにも次のようなインタフェースを使用して、システムにラベルが付けられているかどうかを判別することができます。

- **X** クライアント。マルチレベル機能に依存する X クライアントを書き込んでいる場合は、`XQueryExtension()` ルーチンを使用して `SUN_TSOL` 拡張用の X サーバーをクエリーします。
- シェルスクリプト。システムにラベルが付けられているかどうかを判別するシェルスクリプトを書き込んでいる場合に、`plabel` コマンドを使用します。`plabel(1)` のマニュアルページを参照してください。

  次の例は、`/onnv/onnv-gate/usr/src/cmd/svc/shell/smf_include.sh` スクリプトで使用される `smf_is_system_labeled()` 関数を示します。

```
#
#  Trusted Extensions の別名でも知られるシステムにラベルが付けられている場合、成功を意味するゼロを返す。
#  それ以外の場合は 1。
#
smf_is_system_labeled() {
        [ ! -x /bin/plabel ] && return 1
        /bin/plabel > /dev/null 2>&1;
        return $?
}
```

プロセス機密ラベルへのアクセス

`getplabel()` および `ucred_getlabel()` ルーチンが、プロセスの機密ラベルへのアクセスのために使用されます。次のルーチンの記述には、各ルーチンのプロトタイプ宣言が含まれます。

`int getplabel(m_label_t *label_p);`
  `getplabel()` ルーチンは、呼び出し元プロセスのプロセスラベルを取得します。

  `getplabel(3TSOL)` のマニュアルページを参照してください。

`m_label_t *ucred_getlabel(const ucred_t *uc);`
  `ucred_getlabel()` ルーチンは、遠隔プロセスの資格のラベルを取得します。

  `ucred_getlabel(3C)` のマニュアルページを参照してください。このルーチンの使用例は、57 ページの「`get_peer_label()` ラベル対応関数」を参照してください。

ラベルのためのメモリーの割り当てと解放

`m_label_alloc()`、`m_label_dup()`、および `m_label_free()` ルーチンが、ラベルのためのメモリーの割り当てと解放に使用されます。次のルーチンの記述には、各ルーチンのプロトタイプ宣言が含まれます。

`m_label_t *m_label_alloc(const m_label_type_t label_type);`
`m_label_alloc()` ルーチンは、ヒープの `m_label_t` データ構造にラベルを割り当てます。`getlabel()` や `fgetlabel()` などのルーチンを呼び出す前に、ラベルを割り当てる必要があります。`str_to_label()` などのルーチンでは、`m_label_t` 構造体が自動的に割り当てられます。

`m_label_alloc()` ルーチンを使用してラベルを作成する場合、機密ラベルまたは認可上限ラベルになるようにラベルタイプを設定できます。

`int m_label_dup(m_label_t **dst, const m_label_t *src);`
`m_label_dup()` ルーチンはラベルを複製します。

`void m_label_free(m_label_t *label);`
`m_label_free()` ルーチンは、ラベルに割り当てられたメモリーを解放します。

`m_label_t` 構造体を割り当てる場合、または `m_label_t` 構造体が自動的に割り当てられる別のルーチンを呼び出す場合、割り当てられたメモリーを解放する必要があります。`m_label_free()` ルーチンは、割り当てられたメモリーを解放します。

`m_label(3TSOL)` のマニュアルページを参照してください。

ファイルのラベルの取得と設定

`setflabel()` ルーチン、`getlabel()` システムコール、および `fgetlabel()` システムコールが、ファイルのラベルの取得と設定のために使用されます。次の記述には、ルーチンおよびシステムコールのプロトタイプ宣言が含まれます。

`int setflabel(const char *path, const m_label_t *label_p);`
`setflabel()` ルーチンは、ファイルの機密ラベルを変更します。ファイルの機密ラベルが変わると、新しいラベルに対応するゾーンにファイルが移動します。ファイルは、ほかのゾーンのルートに相対的な新しいパス名に移動します。

`setflabel(3TSOL)` のマニュアルページを参照してください。

たとえば、`setflabel()` ルーチンを使用してファイル `/zone/internal/documents/designdoc.odt` のラベルを `INTERNAL` から `RESTRICTED` に変更すると、ファイルの新しいパスは `/zone/restricted/documents/designdoc.odt` になります。移動先のディレクトリが存在しない場合、ファイルは移動しません。

ファイルの機密ラベルを変更すると、元のファイルは削除されます。移動元と移動先のファイルシステムが同じ基礎となるファイルシステムからループバックマウントされる場合は例外です。この場合、ファイルの名前が変更されます。

プロセスがオブジェクトを作成する場合、オブジェクトは呼び出し元プロセスの機密ラベルを継承します。`setflabel()` ルーチンは、ファイルシステムオブジェクトの機密ラベルをプログラムによって設定します。

ファイルマネージャーアプリケーションおよび setlabel コマンドでは、承認ユーザーが既存のファイルを異なる機密ラベルに移動することができます。setlabel(1) のマニュアルページを参照してください。

int getlabel(const char *path, m_label_t *label_p);
getlabel() システムコールは、*path* によって指定されるファイルのラベルを取得します。ラベルは、割り当てた m_label_t 構造体に格納されます。

getlabel(2) のマニュアルページを参照してください。

int fgetlabel(int fd, m_label_t *label_p);
fgetlabel() システムコールは、ファイル記述子を指定することによって、開かれているファイルのラベルを取得します。

m_label_t 構造体を割り当てる場合、割り当てられたメモリーを m_label_free() ルーチンを使用して解放する必要があります。m_label(3TSOL) のマニュアルページを参照してください。

ラベル範囲の取得

getuserrange() および getdevicerange() ルーチンが、それぞれユーザーとデバイスのラベル範囲を取得するために使用されます。次のルーチンの記述には、各ルーチンのプロトタイプ宣言が含まれます。

m_range_t *getuserrange(const char *username);
getuserrange() ルーチンは、指定したユーザーのラベル範囲を取得します。ユーザーがマルチレベルデスクトップにログインするときに、範囲の下限がラベルとして使用されます。範囲の上限、すなわち認可上限は、ラベル付けされたワークスペースにユーザーが割り当てることができる使用可能なラベルに対する上限として使用されます。

ユーザーのラベル範囲のデフォルト値は label_encodings ファイルで指定します。その値は user_attr ファイルによって上書きできます。

setflabel(3TSOL)、label_encodings(4)、および user_attr(4) のマニュアルページを参照してください。

bl_range_t *getdevicerange(const char *device);
getdevicerange() ルーチンは、ユーザーが割り当て可能なデバイスのラベル範囲を取得します。デバイスにラベル範囲を指定しない場合のデフォルト範囲は、上限が ADMIN_HIGH、下限が ADMIN_LOW です。

list_devices コマンドを使用して、デバイスのラベル範囲を表示できます。

list_devices(1) および getdevicerange(3TSOL) のマニュアルページを参照してください。

ゾーンのラベルへのアクセス

次の関数が、ゾーンのオブジェクトからラベル情報を取得します。次のルーチンの記述には、各ルーチンのプロトタイプ宣言が含まれます。

```
char *getpathbylabel(const char *path, char *resolved_path, size_t bufsize,
const m_label_t *sl);
```

`getpathbylabel()` ルーチンは、すべてのシンボリックリンクを展開して `/./`、`/../` の参照を解決し、余分なスラッシュ (`/`) 文字を削除し、*resolved_path* という名前のバッファーにゾーンパス名を格納します。*bufsize* 変数は、このバッファーのバイト単位のサイズを指定します。その結果得られたパスには、シンボリックリンクの構成要素、または `/./` や `/../` はありません。この関数は大域ゾーンからのみ呼び出せます。

ゾーンパス名は機密ラベル *sl* から相対的なものです。存在しないゾーン名に機密ラベルを指定するには、指定する機密ラベルがプロセス機密ラベルより優位であるか優位でないかに応じて、プロセスが `priv_file_upgrade_sl` または `priv_file_downgrade_sl` 特権を表明する必要があります。

`getpathbylabel(3TSOL)` のマニュアルページを参照してください。

```
m_label_t *getzoneidbylabel(const m_label_t *label);
```

`getzoneidbylabel()` ルーチンは、ラベルが *label* であるゾーンのゾーン ID を返します。このルーチンは、指定したゾーンの状態が少なくとも `ZONE_IS_READY` でなければなりません。呼び出し元プロセスのゾーンは指定したゾーンのラベルより優位であるか、呼び出し元プロセスが大域ゾーンにある必要があります。

`getzoneidbylabel(3TSOL)` のマニュアルページを参照してください。

```
m_label_t *getzonelabelbyid(zoneid_t zoneid);
```

`getzonelabelbyid()` ルーチンは *zoneid* の MAC ラベルを返します。このルーチンは、指定したゾーンの状態が少なくとも `ZONE_IS_READY` でなければなりません。呼び出し元プロセスのゾーンは指定したゾーンのラベルより優位であるか、呼び出し元プロセスが大域ゾーンにある必要があります。

`getzonelabelbyid(3TSOL)` のマニュアルページを参照してください。

```
m_label_t *getzonelabelbyname(const char *zonename);
```

`getzonelabelbyname()` ルーチンは、名前が *zonename* であるゾーンの MAC ラベルを返します。このルーチンは、指定したゾーンの状態が少なくとも `ZONE_IS_READY` でなければなりません。呼び出し元プロセスのゾーンは指定したゾーンのラベルより優位であるか、呼び出し元プロセスが大域ゾーンにある必要があります。

`getzonelabelbyname(3TSOL)` のマニュアルページを参照してください。

```
m_label_t *getzonerootbyid(zoneid_t zoneid);
```

`getzonerootbyid()` ルーチンは *zoneid* のルートパス名を返します。このルーチンは、指定したゾーンの状態が少なくとも `ZONE_IS_READY` でなければなりません。

呼び出し元プロセスのゾーンは指定したゾーンのラベルより優位であるか、呼び出し元プロセスが大域ゾーンにある必要があります。返されるパス名は、呼び出し元ゾーンのルートパスから相対的なものです。

`getzonerootbyid(3TSOL)` のマニュアルページを参照してください。

```
m_label_t *getzonerootbylabel(const m_label_t *label);
```

`getzonerootbylabel()` ルーチンは、ラベルが *label* であるゾーンのルートパス名を返します。このルーチンは、指定したゾーンの状態が少なくとも `ZONE_IS_READY` でなければなりません。呼び出し元プロセスのゾーンは指定したゾーンのラベルより優位であるか、呼び出し元プロセスが大域ゾーンにある必要があります。返されるパス名は、呼び出し元ゾーンのルートパスから相対的なものです。

`getzonerootbylabel(3TSOL)` のマニュアルページを参照してください。

```
m_label_t *getzonerootbyname(const char *zonename);
```

`getzonerootbyname()` ルーチンは *zonename* のルートパス名を返します。このルーチンは、指定したゾーンの状態が少なくとも `ZONE_IS_READY` でなければなりません。呼び出し元プロセスのゾーンは指定したゾーンのラベルより優位であるか、呼び出し元プロセスが大域ゾーンにある必要があります。返されるパス名は、呼び出し元ゾーンのルートパスから相対的なものです。

`getzonerootbyname(3TSOL)` のマニュアルページを参照してください。

遠隔ホストタイプの取得

このルーチンが遠隔ホストタイプを判別します。次のルーチンの記述にはプロトタイプ宣言が含まれます。

```
tsol_host_type_t tsol_getrhtype(char *hostname);
```

`tsol_getrhtype()` ルーチンは、カーネルレベルのネットワーク情報を照会して、指定されたホスト名に関連付けられているホストタイプを判別します。*hostname* は、通常のホスト名、IP アドレス、またはネットワークワイルドカードアドレスです。返される値は、`tsol_host_type_t` 構造体に定義されている列挙型のいずれかです。現在、そのタイプは `UNLABELED` および `SUN_CIPSO` です。

`tsol_getrhtype(3TSOL)` のマニュアルページを参照してください。

ラベルと文字列との変換

`label_to_str()` および `str_to_label()` ルーチンが、ラベルと文字列との変換に使用されます。次のルーチンの記述には、各ルーチンのプロトタイプ宣言が含まれます。

`int label_to_str(const m_label_t *label, char **string, const m_label_str_t conversion_type, uint_t flags);`

`label_to_str()` ルーチンは、ラベル `m_label_t` を文字列に変換します。このルーチンを使用して、格付け名を表示しない文字列にレベルを変換できます。この形式は、共通オブジェクトでの格納に適します。呼び出し元プロセスは変換されるラベルより優位であるか、プロセスに `sys_trans_label` 特権がある必要があります。

`label_to_str(3TSOL)` のマニュアルページを参照してください。

`label_to_str()` ルーチンは、変換される文字列にメモリーを割り当てます。呼び出し元は、`free()` ルーチンを呼び出してこのメモリーを解放する必要があります。

`free(3C)` のマニュアルページを参照してください。

`int str_to_label(const char *string, m_label_t **label, const m_label_type_t label_type, uint_t flags, int *error);`

`str_to_label()` ルーチンは、ラベル文字列をラベル `m_label_t` に変換します。`m_label_t` 構造体を割り当てる場合、割り当てられたメモリーを `m_label_free()` ルーチンを使用して解放する必要があります。

`str_to_label()` ルーチンを使用してラベルを作成する場合、機密ラベルまたは認可上限ラベルになるようにラベルタイプを設定できます。

`str_to_label(3TSOL)` および `m_label(3TSOL)` のマニュアルページを参照してください。

ラベルの読み取り可能バージョン

`label_to_str()` ルーチンは、ラベルの読み取り可能バージョンを提供します。`M_LABEL` 変換タイプは、そのラベルで格付けされている文字列を返します。`M_INTERNAL` 変換タイプは、格付けされていない文字列を返します。格付けされた文字列バージョンは、通常、ウィンドウなどの表示用に使用します。格付けされた文字列は格納に適さない場合があります。印刷のために、いくつかの変換タイプが提供されています。すべての印刷用タイプは、文字列が示すラベルで格付けされる読み取り可能文字列を示します。

conversion_type パラメータはラベル変換のタイプを制御します。次は、conversion_type の有効な値ですが、変換のすべてのタイプが両方のレベルタイプに有効ではありません。

- M_LABEL は、ラベルのタイプ (機密ラベルまたは認可上限) に基づくラベルの文字列です。このラベル文字列はラベルのレベルで格付けされるので、共通オブジェクトで格納する場合は安全でありません。たとえば、CONFIDENTIAL などの M_LABEL 文字列は、ラベルの単語が格付けされることが多いので、共通ディレクトリに格納するのは安全ではありません。
- M_INTERNAL は、格付けされていない表現のラベル文字列です。この文字列は、共通オブジェクトで格納する場合に安全です。たとえば、0x0002-04-48 などの M_INTERNAL 文字列は、LDAP データベースに格納する場合に安全です。
- M_COLOR は、セキュリティー管理者がラベルに関連付けた色を表す文字列です。ラベルと色の関連付けは、label_encodings ファイルの LOCAL DEFINITIONS セクションに格納されます。
- PRINTER_TOP_BOTTOM は、バナーページおよびトレーラページの最上部ラベルと最下部ラベルとして使用される文字列です。
- PRINTER_LABEL は、バナーページのダウングレード警告として使用される文字列です。
- PRINTER_CAVEATS は、バナーページの警告のセクションに使用される文字列です。
- PRINTER_CHANNEL は、バナーページの処理チャネルとして使用される文字列です。

ラベルエンコーディングファイル

label_to_str() ルーチンは label_encodings ファイルのラベル定義を使用します。エンコーディングファイルは、セキュリティー管理者が管理するテキストファイルです。ファイルには、サイト固有のラベルの定義および制約が含まれます。このファイルは /etc/security/tsol/label_encodings に保存されます。label_encodings ファイルについての詳細は、『Solaris Trusted Extensions ラベルの管理』, 『コンパートメントモードワークステーションのラベル作成: エンコード形式』 および label_encodings(4) のマニュアルページを参照してください。

ラベルの比較

blequal()、bldominates()、および blstrictdom() ルーチンが、ラベルの比較のために使用されます。blinrange() ルーチンは、指定されたラベル範囲内にラベルがあるかどうかを判別するために使用されます。このルーチンで、*level* は、格付け、および機密ラベルまたは認可上限ラベルのコンパートメントのセットを指します。

```
int blequal(const blevel_t *level1, const blevel_t *level2);
```

blequal() ルーチンは 2 つのラベルを比較して、*level1* が *level2* と等しいかどうかを判別します。

`int bldominates(const m_label_t *level1, const m_label_t *level2);`
`bldominates()` ルーチンは 2 つのラベルを比較して、*level1* が *level2* より優位であるかどうかを判別します。

`int blstrictdom(const m_label_t *level1, const m_label_t *level2);`
`blstrictdom()` ルーチンは 2 つのラベルを比較して、*level1* が *level2* より厳密に優位であるかどうかを判別します。

`int blinrange(const m_label_t *level, const brange_t *range);`
`blinrange()` ルーチンは、指定された範囲 *range* 内にラベル *level* があるかどうかを判別します。

これらのルーチンは、比較が真の場合はゼロ以外の値、比較が偽の場合は `0` の値を返します。これらのルーチンの詳細は、`blcompare(3TSOL)` のマニュアルページを参照してください。これらのルーチンがマルチレベルの印刷アプリケーションでどのように使用されるかの例は、61 ページの「プリンタのラベル範囲に対するラベル要求の検査」を参照してください。

ラベル関係の詳細は、17 ページの「ラベル関係」を参照してください。

`blmaximum()` および `blminimum()` ルーチンは、指定したラベル範囲の上限および下限を判別するために使用します。

`void blmaximum(m_label_t *maximum_label, const m_label_t *bounding_label);`
`blmaximum()` ルーチンは 2 つのラベルを比較して、範囲の最小上限を見つけます。「最小上限」 は、2 つの認可上限のうちの低い方であり、特定の認可上限のシステムにアクセスしているかどうかを判別するために使用します。

たとえば、このルーチンを使用して、ラベル付けされた 2 つのオブジェクトの情報を組み合わせ、新たに別のオブジェクトを作成してラベル付けするときに使用するラベルを決定します。新しいオブジェクトのラベルは、ラベル付けされた元の 2 つのオブジェクトよりも優位となります。

`blminmax(3TSOL)` のマニュアルページを参照してください。

`void blminimum(m_label_t *minimum_label, const m_label_t *bounding_label);`
`blminimum()` ルーチンは 2 つのラベルを比較して、2 つのレベルによって制限される範囲の最大下限になっているラベルを見つけます。「最大下限」は、2 つのラベルのうちの高い方であり、特定の認可上限のシステムにアクセスできるかどうかを判別するためにも使用します。

`blminmax(3TSOL)` のマニュアルページを参照してください。

機密ラベルの取得

機密ラベルは、ラベル付けされたゾーンおよびほかのプロセスから取得されます。ユーザーは、現在のゾーンの現在の機密ラベルでのみプロセスを起動することができます。

プロセスがオブジェクトを作成する場合、オブジェクトは呼び出し元プロセスの機密ラベルを継承します。`setlabel` コマンドまたは `setflabel()` ルーチンを使用して、ファイルシステムオブジェクトの機密ラベルを設定できます。`setlabel(1)` および `setflabel(3TSOL)` のマニュアルページを参照してください。

次のスクリプト `runwlabel` は、指定するプログラムを、指定するラベル付けされたゾーンで実行します。このスクリプトは大域ゾーンから実行するのでなければなりません。

例 2–1 `runwlabel` スクリプト

`runwlabel` スクリプトは、指定されたプログラムを実行する、ラベル付けされたゾーンの機密ラベルを最初に取得します。スクリプトは、`getzonepath` コマンドを使用して、コマンド行から指定されるラベルからゾーンパスを取得します。`getzonepath(1)` のマニュアルページを参照してください。

次に、`runwlabel` スクリプトは `zoneadm` コマンドを使用して、`getzonepath` によって取得されたゾーンパスに関連付けられているゾーン名を検索します。`zoneadm(1M)` のマニュアルページを参照してください。

最後に、`runwlabel` スクリプトは `zlogin` コマンドを使用して、指定されたラベルに関連付けられているゾーンで、指定されたプログラムを実行します。`zlogin(1)` のマニュアルページを参照してください。

`Confidential:Internal Use Only` ラベルに関連付けられているゾーンで`zonename` コマンドを実行するには、大域ゾーンから `runwlabel` スクリプトを実行します。たとえば、次のように指定します。

```
machine1% runwlabel "Confidential : Internal Use Only" zonename
```

次に `runwlabel` スクリプトのソースを示します。

```
#!/sbin/sh
#
# 使用法:
# runwlabel "my-label" my-program
#
[ ! -x /usr/sbin/zoneadm ] && exit 0    # SUNWzoneu がインストールされていない

PATH=/usr/sbin:/usr/bin; export PATH
```

例 2–1 runwlabel スクリプト (続き)

```
# "my-label" ゾーンに関連付けられているゾーンパスを取得する
# 末尾の "/root" を削除する
zonepath=‘getzonepath "$1" | sed -e ’s/\/root$//’‘
progname="$2"

# このゾーンパスに関連付けられているゾーン名を検索する
for zone in ‘zoneadm list -pi | nawk -F: -v zonepath=${zonepath} ’{
        if ($4 == zonepath) {
            print $2
        }
    }’‘; do

        # 一致するゾーン指定されたコマンドを実行する
        zlogin ${zone} ${progname}
    done
exit
```

次のスクリプト runinzone は、ゾーンが起動されていない場合でも、指定されたゾーンでプログラムを実行します。このスクリプトは大域ゾーンから実行するのでなければなりません。

例 2–2 runinzone スクリプト

このスクリプトは、最初に、指定されたゾーンを起動します。次に、zlogin コマンドを使用して、指定されたゾーンで waitforzone スクリプトを実行します。

waitforzone スクリプトは、ローカルゾーンオートマウンタが起動するのを待機し、指定したプログラムを、指定したユーザーとして実行します。

public ゾーンで /usr/bin/xclock コマンドを実行するには、大域ゾーンから次を実行します。

```
machine1% runinzone public terry /usr/bin/xclock
```

次に runinzone スクリプトのソースを示します。

```
#!/sbin/ksh
zonename=$1
user=$2
program=$3

# 指定されたゾーンを起動する
zoneadm -z ${zonename} boot

# 指定されたゾーンでコマンドを実行する
zlogin ${zonename} /bin/demo/waitforzone ${user} ${program} ${DISPLAY}
```

例 2–2 runinzone スクリプト (続き)

runinzone スクリプトは次のスクリプト waitforzone を呼び出します。

```
#!/bin/ksh
user=$1
program=$2
display=$3

# auto_home トリガーがロードされるのをチェックし
# ローカルゾーンオートマウンタが起動するまで待機する

while [ ! -d /home/${user} ]; do
sleep 1
done

# ここで、指定したコマンドを指定したユーザーとして実行する

su - ${user} -c "${program} -display ${display}"
```

◆ ◆ ◆ 第 3 章

3 ラベルのコーディング例

この章では、第 2 章で説明するラベル API がどのように使用されるかを明らかにするコーディング例を示します。

この章の内容は次のとおりです。

プロセスラベルの取得

このコーディング例は、プログラムが実行されるゾーンの機密ラベルを取得し、出力する方法を示します。

```
#include <tsol/label.h>

main()
{
    m_label_t* pl;
    char *plabel = NULL;
    int retval;

    /* プロセス機密ラベルの m_label_t を割り当てる */
    pl = m_label_alloc(MAC_LABEL);
    /* プロセス機密ラベルを取得する */
    if ((retval = getplabel(pl)) != 0) {
        perror("getplabel(pl) failed");
        exit(1);
```

```
    }

    /* プロセス機密ラベルをテキストに変換し、出力する */
    if ((retval = label_to_str(pl, &plabel, M_LABEL, LONG_NAMES)) != 0) {
        perror("label_to_str(M_LABEL, LONG_NAMES) failed");
        exit(1);
    }
    printf("Process label = %s\n", plabel);

    /* 割り当てられたメモリーを解放する */
    m_label_free(pl);
    free(plabel);
    }
```

`printf()` 文は機密ラベルを出力します。機密ラベルは、プログラムが実行されるゾーンから継承されます。次は、この例のプログラムから出力されたテキストです。

```
Process label = ADMIN_LOW
```

テキスト出力は、`label_encodings` ファイルでの指定に依存します。

ファイルラベルの取得

ファイルの機密ラベルを取得し、そのラベルに対して操作を実行できます。

このコーディング例では、`getlabel()` ルーチンを使用してファイルのラベルを取得します。`fgetlabel()` ルーチンを同様に使用できますが、ファイル記述子に関して機能します。

```
#include <tsol/label.h>

main()
{
    m_label_t* docLabel;
    const char* path = "/zone/restricted/documents/designdoc.odt";
    int retval;
    char* label_string;

    /* ラベルを割り当てて、パスによって指定されるファイルラベルを取得する */
    docLabel = m_label_alloc(MAC_LABEL);
    retval = getlabel(path, docLabel);

    /* ファイルラベルを文字列に変換し、出力する */
    retval = label_to_str(docLabel, &label_string, M_LABEL, LONG_NAMES);
    printf("The file's label = %s\n", label_string);
```

```
    /* 割り当てられたメモリーを解放する */
    m_label_free(docLabel);
    free(label_string);
    }
```

このプログラムを実行した場合の出力は、次のようになります。

```
The file’s label = CONFIDENTIAL : INTERNAL USE ONLY
```

ファイル機密ラベルの設定

ファイルの機密ラベルを変更すると、ファイルの新しいラベルに一致する新しいゾーンにファイルが移動します。

このコーディング例では、プロセスは CONFIDENTIAL ラベルで実行されています。プロセスを実行しているユーザーの認可上限は TOP SECRET です。TOP SECRET ラベルは CONFIDENTIAL ラベルより優位です。このプロセスは機密ラベルを TOP SECRET にアップグレードします。アップグレードを正常に実行するためには、ユーザーに「Upgrade File Label RBAC」承認が必要です。

次のプログラムは upgrade-afile によって呼び出されます。

```
#include <tsol/label.h>

main()
{
   int retval, error;
   m_label_t *fsenslabel;
   char *string = “TOP SECRET”;
   *string1 = “TOP SECRET”;

   /* 新しい機密ラベル値を作成する */
   if ((retval = str_to_label(string, &fsenslabel, MAC_LABEL, L_DEFAULT, &err)) != 0) {
        perror("str_to_label(MAC_LABEL, L_DEFAULT) failed");
        exit(1);
    }

   /* ファイルラベルを新しい値に設定する */
   if ((retval = setflabel(“/export/home/zelda/afile”, &fsenslabel)) != 0) {
        perror("setflabel(“/export/home/zelda/afile”) failed");
        exit(1);
    }

   m_label_free(fsenslabel);
}
```

このプログラムの実行結果は、プロセスに渡されたファイルのラベルを基準にしたプロセスのラベルに依存します。

このプログラムを実行する前後に getlabel コマンドを使用してファイルのラベルを確認します。次に示すように、プログラム実行前の afile のラベルは CONFIDENTIAL です。プログラム実行後の afile のラベルは TOP SECRET です。

```
% pwd
/export/home/zelda
% getlabel afile
afile: CONFIDENTIAL
% update-afile
% getlabel afile
afile: TOP SECRET
```

ファイルを再格付けしたあとに CONFIDENTIAL とラベル付けされたウィンドウから getlabel コマンドを実行すると、ファイルが表示されなくなります。TOP SECRET とラベル付けされたウィンドウから getlabel コマンドを実行すると、再格付けされたファイルが表示されます。

2 つのラベル間の関係の判別

アプリケーションが、異なる機密ラベルでデータにアクセスする場合、アクセス操作を許可する前に、コードを検査してプロセスラベルとデータラベルの関係が正しいことを確認します。アクセスされるオブジェクトの機密ラベルを検査して、システムによってアクセスが許可されるかどうかを判別します。

次のコーディング例は、2 つの機密ラベルの等位、優位、および厳密優位を検査する方法を示します。プログラムは、ファイルのラベルに対してプロセスのラベルが優位であるか、または等位であるかを検査します。

```
#include <stdio.h>
#include <stdlib.h>

#include <tsol/label.h>

main(int argc, char *argv[])
{
   m_label_t *plabel;
   m_label_t *flabel;

   plabel = m_label_alloc(MAC_LABEL);
   flabel = m_label_alloc(MAC_LABEL);

   if (getplabel(plabel) == -1) {
      perror("getplabel");
```

```
        exit(1);
    }
    if (getlabel(argv[1], flabel) == -1) {
        perror("getlabel");
        exit(1);
    }

    if (blequal(plabel, flabel)) {
        printf("Labels are equal\n");
    }
    if (bldominates(plabel, flabel)) {
        printf("Process label dominates file label\n");
    }
    if (blstrictdom(plabel, flabel)) {
        printf("Process label strictly dominates file label\n");
    }

    m_label_free(plabel);
    m_label_free(flabel);

    return (0);
}
```

このプログラムによって出力されるテキストは、次のように、プロセスに渡されたファイルのラベルを基準にしたプロセスのラベルに依存します。

- "dominates" は "equal" を含むので、ラベルが等位である場合の出力は、次のようになります。

```
Labels are equal
Process label dominates file label
```

- プロセスのラベルがファイルのラベルより厳密に優位である場合の出力は、次のようになります。

```
Process label strictly dominates file label
```

ラベルのカラー名の取得

このコーディング例では、`label_to_str()` 関数を使用してラベルのカラー名を取得します。カラー名とラベルのマッピングは `label_encodings` ファイルで定義します。

```
#include <stdlib.h>
#include <stdio.h>

#include <tsol/label.h>
```

```
int
main()
{
   m_label_t *plabel;
   char *label = NULL;
   char *color = NULL;

   plabel = m_label_alloc(MAC_LABEL);

   if (getplabel(plabel) == -1) {
      perror("getplabel");
      exit(1);
   }

   if (label_to_str(plabel, &color, M_COLOR, 0) != 0) {
      perror("label_to_string(M_COLOR)");
      exit(1);
   }
   if (label_to_str(plabel, &label, M_LABEL, DEF_NAMES) != 0) {
      perror("label_to_str(M_LABEL)");
      exit(1);
   }

   printf("The color for the \"%s\" label is \"%s\".\n, label, color);

   m_label_free(plabel);

   return (0);
}
```

label_encodings ファイルで青がラベル CONFIDENTIAL にマップされている場合、プログラムの出力は次のようになります。

```
The color for the "CONFIDENTIAL" label is "BLUE".
```

プリンタバナー情報の取得

label_encodings ファイルには、プリンタ出力としてセキュリティー情報を印刷する場合に役立ついくつかの変換を定義します。ラベル変換は、ページの最上部と最下部に出力されます。処理チャネルなどのその他の変換は、バナーページに表示されます。

次のコーディング例では、label_to_str() ルーチンが、ラベルをヘッダーとフッター、警告セクション、処理チャネルなどの文字列に変換します。このルーチンは、第 4 章に示すように、Trusted Extensions 印刷システムによって内部的に使用されます。

```
#include <stdlib.h>
#include <stdio.h>

#include <tsol/label.h>

int
main()
{
   m_label_t *plabel;
   char *header = NULL;
   char *label = NULL;
   char *caveats = NULL;
   char *channels = NULL;

   plabel = m_label_alloc(MAC_LABEL);
   if (getplabel(plabel) == -1) {
      perror("getplabel");
      exit(1);
   }
   if (label_to_str(plabel, &header, PRINTER_TOP_BOTTOM, DEF_NAMES) != 0) {
      perror("label_to_str: header");
      exit(1);
   }
   if (label_to_str(plabel, &label, PRINTER_LABEL, DEF_NAMES) != 0) {
      perror("label_to_str: label");
      exit(1);
   }
   if (label_to_str(plabel, &caveats, PRINTER_CAVEATS, DEF_NAMES) != 0) {
      perror("label_to_str: caveats");
      exit(1);
   }
   if (label_to_str(plabel, &channels, PRINTER_CHANNELS, DEF_NAMES) != 0) {
      perror("label_to_str: channels");
      exit(1);
   }

   printf("\t\t\t\"%s\"\n\n", header);
   printf("\t\tUnless manually reviewed and downgraded, this output\n");
   printf("\t\tmust be protected at the following label:\n\n");
   printf("\t\t\t\"%s\"\n", label);
   printf("\n\n\n");
   printf("\t\t\"%s\"\n", caveats);
   printf("\t\t\"%s\"\n", channels);
   printf("\n\n");
   printf("\t\t\t\"%s\"\n", header);

   m_label_free(plabel);
```

```
   return (0);
}
```

TS SA SB のプロセスラベルの場合、テキスト出力は次のようになります。

```
            "TOP SECRET"

        Unless manually reviewed and downgraded, this output
        must be protected at the following label:

            "TOP SECRET A B SA SB"



        "(FULL SB NAME) (FULL SA NAME)"
        "HANDLE VIA (CH B)/(CH A) CHANNELS JOINTLY"


            "TOP SECRET"
```

詳細は、label_encodings(4) のマニュアルページ、『コンパートメントモードワークステーションのラベル作成: エンコード形式』、および『Solaris Trusted Extensions ラベルの管理』を参照してください。

◆ ◆ ◆ 第 4 章

4 印刷とラベル API

印刷は、ラベル対応でなければならないサービスの 1 つです。この章では、Trusted Extensions 用に開発されたマルチレベル印刷サービスの例を使用することによって Solaris Trusted Extensions ラベル API について説明します。

この章の内容は次のとおりです。

ラベル付けされた出力の印刷

通常、プリンタは共有リソースです。マルチレベル印刷によって、異なるセキュリティーレベルで作業するユーザーは、セキュリティーポリシーの制限に従いながら、プリンタを共有できます。印刷サービスもラベル対応なので、印刷されるドキュメントにラベルをはっきりと表記できます。

出力がラベル付けされるようにプリンタを構成するには、役割ベースアクセス制御 (RBAC) でシステム管理者役割になります。印刷ジョブが開始されるセッションラベルは、バナーページおよびトレーラページに印刷されます。セッションのラベルは、印刷されるすべてのページのヘッダーおよびフッターにも追加されます。ラベルは印刷アダプタによって印刷できます。Trusted Extensions 印刷アダプタは、印刷要求が開始されたホストラベルまたはゾーンラベルを特定します。アダプタは印刷ジョブとともにこのラベル情報を渡すので、印刷される出力のラベル付けが可能になります。

ラベル対応アプリケーションの設計

ほとんどのアプリケーションはラベル対応である必要はありません。したがって、大部分の Solaris ソフトウェアアプリケーションは、変更せずに Trusted Extensions で実行されます。Trusted Extensions のラベルベースアクセス制限は、Solaris OS 標準と整合して機能するように設計されています。マルチレベルポートにバインドするプロセスは、複数のラベルでデータを受信し、信頼されてセキュリティーポリシーが強制されるので、一般に、ラベル対応である必要があります。

たとえば、アプリケーションが必要なリソースより低位のラベルで実行されているために、アプリケーションがリソースにアクセスできないことがあります。ただし、そのリソースへアクセスしようとすることは特別なエラー条件にはなりません。代わりに、アプリケーションによって「ファイルが見つかりませんでした (`File not found`)」エラーが表示されます。あるいは、アクセスが許可されない高位のラベルを持つ情報にアプリケーションがアクセスしようとすることがあります。それに対し、セキュリティーポリシーでは、十分な特権がないアプリケーションはより高位のラベルを持つリソースの存在を認識できないとされています。そのため、アプリケーションのラベルより高位のラベルを持つリソースにアプリケーションがアクセスしようとした場合、そこで発生するエラー条件はラベルに固有ではありません。そのエラーメッセージは、存在しないリソースにアクセスしようとするアプリケーションに返されるエラーメッセージと同じです。「特別なエラー条件」がないことが、セキュリティーの原則の強制に役立ちます。

Trusted Extensions では、アプリケーションではなく、オペレーティングシステムがセキュリティーポリシーを強制します。このセキュリティーポリシーは「必須アクセス制御 *(MAC)* ポリシー」と呼ばれます。たとえば、アプリケーションは、保護されているリソースにアクセス可能であるか否かを判別しません。最終的にオペレーティングシステムが MAC ポリシーを強制します。リソースにアクセスするのに十分な特権をアプリケーションが持たない場合、そのリソースはアプリケーションで使用できません。そのため、アプリケーションは、ラベル付けされたリソースにアクセスするためにラベルに関して何も知る必要はありません。

同様に、ほとんどのラベル対応アプリケーションは、ラベル対応でないアプリケーションと一貫して動作するように設計する必要があります。ラベル対応のアプリケーションは、単一ラベルのみを含む環境、ラベル付けされていない環境、および複数のラベルを含む環境で基本的に同じ動作をする必要があります。単一ラベル環境の例は、特定ラベルのユーザーセッションが同じラベルでデバイスをマウントする場合です。「ラベル付けされていない環境」では、ラベルは明示的に設定されず、デフォルトラベルが `tnrhdb` データベースで指定されます。`smtnrhdb(1M)` のマニュアルページを参照してください。

マルチレベル印刷サービスについて

印刷サービスは異なるラベルで動作するプロセスからの要求を受け入れるので、印刷はラベル対応である必要があります。通常、MAC は、ユーザーが作業しているのと同じラベルのリソースに対してのみアクセスを許可します。印刷要求が同じラベルでのみ発行される場合でも、印刷される出力の印刷ページにラベルを表示できるように、印刷はラベル対応にすべきです。

ラベルを処理するために、印刷サービスは次の基本機能を実行する必要があります。

- 印刷プロセスが実行されているホストがラベル付けされているか否かを判別する
- 印刷プロセスがラベル付けされている環境で実行されている場合、印刷要求が開始するネットワーク接続の資格を取得する (資格にはそのプロセスのラベルが含まれます)
- ネットワーク資格からラベルを抽出する
- プリンタのラベル範囲、すなわち、プリンタが要求を受け入れることができるラベルの範囲を取得する
- 指定されたプリンタのラベルの受け入れ可能な範囲内にユーザーのラベルがあるか否かを判別する

get_peer_label() ラベル対応関数

lp/lib/lp/tx.c ファイルの get_peer_label() 関数は、Trusted Extensions におけるマルチレベル印刷のロジックを実装します。これ以降の各節でこの関数について説明し、順を追ってそれを実装します。

Trusted Extensions ソフトウェアでは、印刷サービスにおけるラベル処理のための多くのロジックは get_peer_label() 関数にあります。この関数は、ucred_t データ構造体の遠隔プロセスに関する資格を取得し、その資格からラベルを抽出します。

次は get_peer_label() のコードです。

```
int
get_peer_label(int fd, char **slabel)
{
    if (is_system_labeled()) {
        ucred_t *uc = NULL;
        m_label_t *sl;
        char *pslabel = NULL; /* peer's slabel */

        if ((fd < 0) || (slabel == NULL)) {
            errno = EINVAL;
            return (-1);
```

```
        }

        if (getpeerucred(fd, &uc) == -1)
            return (-1);

        sl = ucred_getlabel(uc);
        if (label_to_str(sl, &pslabel, M_INTERNAL, DEF_NAMES) != 0)
            syslog(LOG_WARNING, "label_to_str(): %m");
        ucred_free(uc);

        if (pslabel != NULL) {
            syslog(LOG_DEBUG, "get_peer_label(%d, %s): becomes %s",
                fd, (*slabel ? *slabel : "NULL"), pslabel);
            if (*slabel != NULL)
                free(*slabel);
            *slabel = strdup(pslabel);
        }
    }

    return (0);
}
```

ラベル付けされている環境で印刷サービスが実行されているか否かの判別

印刷サービスは、ラベル付けされている環境とラベル付けされていない環境で機能するように設計されています。そのため、印刷アプリケーションは、遠隔ホストのラベルが要求されるときと、そのラベルが適用されるか否かを決定する必要があります。印刷プロセスは、最初、それ自身の環境を検査します。プロセスがラベル対応の環境で実行されているか否かを確認します。

アプリケーションは、遠隔要求がラベル付けされているか否かを最初に判別しません。その代わりに、印刷アプリケーションはそれ自身の環境がラベル付けされているか否かを判別します。アプリケーションがラベル付けされているホストで実行されていない場合、MAC ポリシーによって、印刷アプリケーションはラベル付けされた要求を受信できません。

印刷サービスは `is_system_labeled()` 関数を使用して、ラベル付けされた環境でプロセスが実行されているか否かを判別します。この関数についての詳細は、`is_system_labeled(3C)` のマニュアルページを参照してください。

次のコード抜粋は、ラベル付けされた環境でアプリケーションが実行されているか否かをどのように判別するかを示します。

```
if (is_system_labeled()) {
    ucred_t *uc = NULL;
```

```
    m_label_t *sl;
    char *pslabel = NULL; /* peer's slabel */

    if ((fd < 0) || (slabel == NULL)) {
        errno = EINVAL;
        return (-1);
    }
```

Trusted Extensions が設定されたシステムで印刷アダプタプロセスが実行されている場合、is_system_labeled() 関数は遠隔プロセスから ucred_t 資格抽象を取得します。次に、遠隔プロセスの ucred_t データ構造体およびピアのラベルが NULL に設定されます。資格およびピアのラベルの値を返す関数がデータ構造体を満たします。そのデータ構造体については、次の節で説明します。

get_peer_label() ルーチン全体のソースは、57 ページの「get_peer_label() ラベル対応関数」を参照してください。

遠隔ホスト資格について

Solaris OS ネットワーク API は、プロセスの資格の抽象を提供します。この資格データは、ネットワーク接続を通じて使用できます。資格は、Solaris 10 リリースで導入された ucred_t データ構造体によって表現されます。この構造体はプロセスのラベルを含むことができます。

ucred API には、遠隔プロセスから ucred_t データ構造体を取得するための関数があります。この API は、ucred_t データ構造体からラベルを抽出するための関数も提供します。

資格と遠隔ホストラベルの取得

遠隔プロセスのラベルの取得は、2 ステップの手続きで実行されます。最初に、資格を取得します。次に、その資格からラベルを取得します。

資格は、遠隔プロセスの ucred_t データ構造体にあります。ラベルは、資格の m_label_t データ構造体にあります。遠隔プロセスの資格を取得したあとに、その資格からラベル情報を抽出します。

getpeerucred() 関数が、遠隔プロセスから ucred_t 資格データ構造体を取得します。ucred_getlabel() 関数が、ucred_t データ構造体からラベルを抽出します。get_peer_label() 関数では、次のように 2 ステップの手続きがコーディングされています。

```
if (getpeerucred(fd, &uc) == -1)
    return (-1);

sl = ucred_getlabel(uc);
```

get_peer_label() ルーチン全体のソースは、57 ページの「get_peer_label() ラベル対応関数」を参照してください。

この 2 つの関数については、getpeerucred(3C) および ucred_getlabel(3C) のマニュアルページを参照してください。

遠隔ホストのラベルのほかに、遠隔ホストのタイプも取得できます。遠隔ホストタイプを取得するには、tsol_getrhtype() ルーチンを使用します。39 ページの「遠隔ホストタイプの取得」を参照してください。

label_to_str() 関数の使用

資格および遠隔ホストラベルを取得したあとに、アプリケーションは label_to_str() を使用してラベルデータ構造体を文字列に変換できます。ラベルデータ構造体の文字列形式は、アプリケーションによって使用できます。

Trusted Extensions 印刷サービスでは、ラベルは文字列として返されます。get_peer_label() 関数は、m_label_t データ構造体で label_to_str() を呼び出すことによって取得される文字列を返します。この文字列値は、get_peer_label() 関数の slabel パラメータ、char** slabel に返されます。

次のコード抜粋は、label_to_str() 関数がどのように使用されるかを示します。

```
sl = ucred_getlabel(uc);
if (label_to_str(sl, &pslabel, M_INTERNAL, DEF_NAMES) != 0)
    syslog(LOG_WARNING, "label_to_str(): %m");
ucred_free(uc);

if (pslabel != NULL) {
    syslog(LOG_DEBUG, "get_peer_label(%d, %s): becomes %s",
        fd, (*slabel ? *slabel : "NULL"), pslabel);
    if (*slabel != NULL)
        free(*slabel);
    *slabel = strdup(pslabel);
}
```

get_peer_label() ルーチン全体のソースは、57 ページの「get_peer_label() ラベル対応関数」を参照してください。

メモリー管理の処理

57 ページの「get_peer_label() ラベル対応関数」に示すように、多くの場合、ラベルは動的に割り当てられます。関数 str_to_label()、label_to_str()、getdevicerange()、およびその他の関数は、呼び出し元によって解放されなければならないメモリーを割り当てます。これらの関数の次のマニュアルページに、メモリー割り当ての要件が説明されています。

- getdevicerange(3TSOL)
- label_to_str(3TSOL)
- m_label(3TSOL)
- str_to_label(3TSOL)

返されたラベル文字列の使用

get_peer_label() 関数は、遠隔ホストからラベルを抽出し、そのラベルを文字列として返します。典型的なラベル対応アプリケーションと同様に、印刷アプリケーションは次の目的でラベルを使用します。

- ラベルに関連付けられている情報が、正しいラベルではっきりとマークされていることを確認するため。バナーページとトレーラページ、およびヘッダーとフッターは、印刷されるドキュメントのラベルでマークされます。
- ラベル付けされた別のリソースによって特定操作が実行されるのを、あるリソースのラベルが許可するのを検査するため。すなわち、要求プロセスのラベルは、その要求プロセスからの要求をプリンタが受け入れるのを許可します。この許可は、プリンタに割り当てられているラベルの範囲に基づきます。

プリンタのラベル範囲に対するラベル要求の検査

印刷アプリケーションには、ラベルを検査するためのコードが lp/cmd/lpsched/validate.c ファイルに含まれています。

アプリケーションのタイプによっては、2 つの特定ラベルを比較する必要があります。たとえば、アプリケーションは、あるラベルが別のラベルより厳密に優位であるか否かを判別しなければならないことがあります。このようなアプリケーションは、2 つのラベルを比較するための API 関数を使用します。

ただし、印刷アプリケーションは、ラベルの範囲を基準にしています。プリンタは、異なるラベルの範囲からの印刷要求を受け入れるように構成されます。そのため、印刷アプリケーションは、ラベルと範囲を照合する API 関数を使用します。アプリケーションは、遠隔ホストのラベルがプリンタによって許可されるラベルの範囲内にあることを検査します。

validate.c ファイルでは、印刷アプリケーションは blinrange() 関数を使用して、遠隔ホストのラベルとプリンタのラベル範囲を照合します。この検査は、次に示すように tsol_check_printer_label_range() 関数内で実行されます。

```
static int
tsol_check_printer_label_range(char *slabel, const char *printer)
{
    int            in_range = 0;
    int            err = 0;
    blrange_t        *range;
    m_label_t    *sl = NULL;

    if (slabel == NULL)
        return (0);

    if ((err =
        (str_to_label(slabel, &sl, USER_CLEAR, L_NO_CORRECTION, &in_range)))
        == -1) {
        /* プリンタ最大ラベルに対する str_to_label エラー */
        return (0);
    }
    if ((range = getdevicerange(printer)) == NULL) {
        m_label_free(sl);
        return (0);
    }

    /* blinrange は範囲内には 真 (1)、範囲外には偽 (0) を返す */
    in_range = blinrange(sl, range);

    m_label_free(sl);
    m_label_free(range->lower_bound);
    m_label_free(range->upper_bound);
    free(range);

    return (in_range);
}
```

tsol_check_printer_label_range() 関数は、get_peer_label() 関数によって返されるラベル、およびプリンタ名をパラメータとして受け取ります。

ラベルを比較する前に、tsol_check_printer_label_range() が str_to_label() 関数を使用して文字列をラベルに変換します。

ラベルタイプが USER_CLEAR に設定され、これによって、関連付けられているオブジェクトの認可上限ラベルが生成されます。認可上限ラベルは、blinrange() 関数が実行する範囲検査で適切なレベルのラベルが使用されるようにします。

str_to_label() から取得される sl ラベルが検査されることによって、要求されるデバイスであるプリンタの範囲内に遠隔ホストのラベル slabel があるか否かを判別します。このラベルはプリンタのラベルと照合されます。プリンタの範囲は、選択したプリンタに対して getdevicerange() 関数を呼び出すことによって取得されます。範囲は blrange_t データ構造体として返されます。

blrange_t データ構造体のプリンタのラベル範囲は、要求元の認可上限ラベルとともに blinrange() 関数に渡されます。blinrange(3TSOL) のマニュアルページを参照してください。

次のコード抜粋は、validate.c ファイルの _validate() 関数を示します。この関数は、印刷要求を処理するためのプリンタの検索に使用されます。このコードは、要求に関連付けられているユーザー ID およびラベルを、各プリンタに関連付けられている許可ユーザーおよびラベル範囲のセットと照合します。

```
/*
 * 1 台のプリンタが指定された場合、要求はそのプリンタと照合される。
 * もっとも有用な情報をユーザーに示すことができるように、あとで
 * 受け入れ/拒否検査を実行する。
 */
if (pps) {
    (pc = &single)->pps = pps;

    /* プリンタはユーザーへのアクセスを許可するか? */
    if (!CHKU(prs, pps)) {
        ret = MDENYDEST;
        goto Return;
    }

    /* プリンタラベル範囲を検査する */
    if (is_system_labeled() && prs->secure->slabel != NULL) {
        if (tsol_check_printer_label_range(prs->secure->slabel,
            pps->printer->name) == 0) {
            ret = MDENYDEST;
            goto Return;
        }
    }
```

◆ ◆ ◆ 第 5 章

プロセス間通信

Solaris Trusted Extensions が構成されているシステムは、必須アクセス制御 (MAC) および任意アクセス制御 (DAC) を実行します。アクセス制御は、同じホストでのプロセス間通信、およびネットワークをまたがるプロセス間通信で実行されます。この章では、Trusted Extensions が設定されているシステムで使用できるプロセス間通信 (IPC) のメカニズムについて概説します。さらに、アクセス制御がどのように適用されるかについても説明します。

Trusted Extensions API を Solaris OS で使用する方法の例は、Solaris ソースコードを参照してください。OpenSolaris の Web サイト (`http://jp.opensolaris.org/`) の左のナビゲーションバーにある「Source Browser」をクリックします。Source Browser を使用して Solaris のソースコードを検索します。

この章の内容は次のとおりです。

マルチレベルポートについて

Solaris Trusted Extensions が構成されているシステムは、単一レベルポートおよびマルチレベルポートに対応します。これらのポートは、アプリケーション間を接続するために使用されます。マルチレベルポートは、ポートに定義されている機密ラベルの範囲内のデータを受信できます。単一レベルポートは、指定された機密ラベルのデータのみを受信できます。

- 単一レベルポート – 通信チャネルは、2 つの非特権アプリケーション間に確立されます。通信終端の機密ラベルは等位でなければなりません。
- マルチレベルポート – 通信チャネルは、`net_bindmlp` 特権が実効セットであるアプリケーションと、異なる機密ラベルで実行される不定数の非特権アプリケーションとの間に確立されます。プロセスの実効セットに `net_bindmlp` 特権を持つ

アプリケーションは、受信側アプリケーションの機密ラベルに関係なく、アプリケーションからのすべてのデータを受信できます。

マルチレベルポートは、異なるラベルで実行されている 2 つの Trusted Extensions アプリケーション間で接続を確立するためのサーバー側のメカニズムです。Trusted Extensions クライアントアプリケーションで、ラベルの異なる信頼できないオペレーティングシステムで実行されるサービスと通信する場合、`SO_MAC_EXEMPT` ソケットオプションを使用することもできます。詳細は、26 ページの「MAC 適用外ソケット」を参照してください。

注意 – 接続がマルチレベルである場合、アプリケーションの接続が 1 つの機密ラベルで行われ、データの送信または受信が別の機密ラベルで行われるのではないようにしてください。このような構成では、承認されていない宛先にデータが到達することがあります。

トラステッドネットワークライブラリは、パケットからラベルを取得するインタフェースを提供します。ネットワークパケットのプログラム化された操作は必要ありません。ただし、メッセージを送信する前に、メッセージのセキュリティー属性を変更することはできません。また、メッセージが送信される通信終端でもセキュリティー属性を変更できません。パケットのラベルの読み取りは、パケットのその他のセキュリティー情報と同様に行えます。`ucred_getlabel()` 関数を使用してラベル情報を取得します。

アプリケーションがマルチレベルポートの使用を必要とする場合、そのポートはプログラムでは作成できません。アプリケーション用のマルチレベルポートの作成は、システム管理者に依頼する必要があります。

マルチレベルポートについての詳細は、次を参照してください。

- 『Solaris Trusted Extensions 管理の手順』の「ゾーンとマルチレベルポート」
- 『Solaris Trusted Extensions 管理の手順』の「ゾーンにマルチレベルポートを作成する」
- 『Solaris Trusted Extensions 管理の手順』の「マルチレベルプリンタサーバーとそのプリンタを構成する」

通信終端

Trusted Extensions ソフトウェアは、次のソケットベースのメカニズムを使用することによって、通信終端間の IPC に対応します。

- バークレーソケット
- トランスポートレイヤーインタフェース (TLI)
- 遠隔手続き呼び出し (RPC)

この節では、ソケット通信メカニズムとそれに関連するセキュリティーポリシーについて概説します。セキュリティーポリシーと適用可能な特権については、該当するマニュアルページを参照してください。

これらのメカニズムのほかに、Trusted Extensions はマルチレベルポートにも対応します。65 ページの「マルチレベルポートについて」を参照してください。

バークレーソケットと TLI

Trusted Extensions ソフトウェアは、単一ポートおよびマルチレベルポートでバークレーソケットおよび TLI を使用することによって、ネットワーク通信に対応します。システムコールの AF_UNIX ファミリは、完全解決のパス名で指定される特別なファイルを使用して、同じラベル付けされたゾーンでプロセス間接続を確立します。システムコールの AF_INET ファミリは、IP アドレスとポート番号を使用して、ネットワークにまたがるプロセス間接続を確立します。

AF_UNIX ファミリ

インタフェースの AF_UNIX ファミリでは、UNIX® ドメインソケットである特別な 1 つのファイルに対してサーバーバインドを 1 つだけ確立できます。AF_UNIX ファミリはマルチレベルポートに対応しません。

UNIX ドメインソケット同様に、ドアおよび名前付きパイプは、認識し合う目的で特別なファイルを使用します。

すべての Trusted Extensions IPC メカニズムのデフォルトポリシーは、すべてのメカニズムが 1 つのラベル付けされたゾーン内で機能するように制約されることです。このポリシーの例外は次のとおりです。

- 大域ゾーン管理者は、ラベルが所有ゾーンより優位であるゾーンに対して名前付きパイプ (FIFO) を使用可能にできます。管理者は、FIFO を含むディレクトリをループバックマウントすることによってこれを行います。

  より高いレベルのゾーンで実行されるプロセスは、読み取り専用モードで FIFO を開くことができます。プロセスは下位書き込みのために FIFO を使用できません。
- 大域ゾーンランデブファイルがラベル付けされたゾーンにループバックマウントされる場合、ラベル付けされたゾーンは大域ゾーンドアサーバーにアクセスできます。

  Trusted Extensions ソフトウェアは、labeld および nscd のドアベースサービスをサポートするドアポリシーに従います。デフォルトの zonecfg テンプレートは、大域ゾーンの /var/tsol/doors ディレクトリが、ラベル付けされたゾーンのそれぞれにループバックマウントされることを指定します。

AF_INET ファミリ

AF_INET ファミリでは、プロセスは、特権または非特権のポート番号に対して単一ラベル接続またはマルチラベル接続を確立できます。特権ポート番号に接続するには、net_priv_addr 特権が必要です。マルチレベルポート接続にするには、net_bindmlp 特権も必要です。

サーバープロセスには、マルチレベルポート接続のために、その実効セットに net_bindmlp 特権が必要です。それに対して単一レベルポート接続を行う場合、サーバープロセスにはソケットに対する必須同等読み取りアクセスが必要であり、クライアントプロセスには必須同等書き込みアクセスが必要です。両方のプロセスには、ファイルに対する必須と任意のアクセスが必要です。ファイルへのアクセスが拒否される場合、アクセスが拒否されるプロセスには、アクセスを得るために、適切なファイル特権がその実効セットに必要です。

次のコーディング例は、接続されたクライアントのラベルをマルチレベルサーバーがどのように取得するかを示します。標準 C ライブラリ関数 getpeerucred() が、接続されたソケットまたは STREAM ピアの資格を取得します。Trusted Extensions において、マルチレベルポートサーバーの待機ソケットが接続要求を受け入れると、通常、最初の引数がクライアントソケットファイル記述子になります。Trusted Extensions アプリケーションは、標準のアプリケーションプログラムとまったく同じに getpeerucred() 関数を使用します。Trusted Extensions には ucred_getlabel() が追加され、ラベルを返します。詳細は、ucred_get(3C) のマニュアルページを参照してください。

```
/*
 * この例は、接続されたクライアントのラベルをマルチレベルサーバーが
 * どのように取得するかを示す。
 */
void
remote_client_label(int svr_fd)
{
    ucred_t *uc = NULL;
    m_label_t *sl;
    struct sockaddr_in6 remote_addr;

    bzero((void *)&remote_addr, sizeof (struct sockaddr_in6));

    while (1) {
        int clnt_fd;
        clnt_fd = accept(svr_fd, (struct sockaddr *)&remote_addr,
                &sizeof (struct sockaddr_in6));

        /*
         * ソケットからクライアント属性を取得する
         */
        if (getpeerucred(clnt_fd, &uc) == -1) {
```

```
            return;
        }

        /*
         * ucred 構造体から各フィールドを抽出する
         */

        sl = ucred_getlabel(uc);

        /*
         * セキュリティーラベルの使用法
         * .....
         */

        ucred_free(uc);
        close(clnt_fd);
    }
}
```

RPC メカニズム

Trusted Extensions ソフトウェアは、遠隔手続き呼び出し (RPC) に対するマルチレベルポートに対応します。クライアントアプリケーションは、特定サービスが使用可能かどうかの問い合わせをサーバーの PORTMAPPER サービス (ポート 111) に対して送信できます。要求されたサービスがサーバーの PORTMAPPER に登録されている場合、サーバーは匿名ポートを動的に割り当て、そのポートをクライアントに返します。

Solaris Trusted Extensions システムでは、管理者が PORTMAPPER ポートをマルチレベルポートとして構成できるので、複数の単一レベルアプリケーションでこのサービスを使用できます。PORTMAPPER ポートをマルチレベルポートにすると、PORTMAPPER サービスによって割り当てられるすべての匿名ポートもマルチレベルポートになります。これ以外の、匿名のマルチレベルポートを制御するための、プログラム可能なインタフェースも管理インタフェースもありません。

UDP とマルチレベルポートの使用

前の節で説明した PORTMAPPER サービスは、UDP を使用して実装されます。TCP と異なり、UDP ソケットは接続指向ではないので、マルチレベルポートでクライアントに応答する場合にどの資格を使用するかに関してあいまいになることがあります。したがって、クライアントの要求ソケットは、サーバーの応答ソケットに明示的に関連付けられなければなりません。これを関連付けるには、SO_RECVUCRED ソケットオプションを使用します。

SO_RECVUCRED が UDP ソケットで設定されると、カーネル UDP モジュールは ucred 構造体のラベルを補助データとしてアプリケーションに渡します。ucred の level と type の値がそれぞれ、SOL_SOCKET と SCM_UCRED です。

アプリケーションは次のいずれかの方法でこの ucred 構造体を処理します。

- この ucred 構造体を受信バッファーから送信バッファーにコピーする
- 受信バッファーを送信バッファーとして再利用し、受信バッファーの ucred 構造体をそのままにする

次のコード抜粋は再利用の場合を示します。

```
/*
 * src で SCM_UCRED を検索し、dest のみのオプションに
 * ポインタを指定する。これら 2 つの 'netbuf' 構造体が同じ
 * である場合もある。そのため、コードで dest の変更後に
 * src を照会するときに注意が必要。
 */
static void
extract_cred(const struct netbuf *src, struct netbuf *dest)
{
    char *cp = src->buf;
    unsigned int len = src->len;
    const struct T_opthdr *opt;
    unsigned int olen;

    while (len >= sizeof (*opt)) {
        /* LINTED: pointer alignment */
        opt = (const struct T_opthdr *)cp;
        olen = opt->len;
        if (olen > len || olen < sizeof (*opt) ||
            !IS_P2ALIGNED(olen, sizeof (t_uscalar_t)))
            break;
        if (opt->level == SOL_SOCKET &&
            opt->name == SCM_UCRED) {
            dest->buf = cp;
            dest->len = olen;
            return;
        }
        cp += olen;
        len -= olen;
    }
    dest->len = 0;
}
```

次のコード抜粋は、受信バッファーからユーザー資格にアクセスする方法を示します。

```
void
examine_udp_label()
{
    struct msghdr   recv_msg;
    struct cmsghdr  *cmsgp;
    char message[MAX_MSGLEN+1];
    char inmsg[MAX_MSGLEN+1];
    int on = 1;

    setsockopt(sockfd, SOL_SOCKET, SO_RECVUCRED, (void *)&on,
        sizeof (int));

    [...]

    while (1) {
       if (recvmsg(sockfd, &recv_msg, 0) < 0) {
           (void) fprintf(stderr, "recvmsg_errno:   %d\n", errno);
           exit(1);
           }

           /*
            * 補助データで ucred を検査する
            */
           ucred = NULL;

           for (cmsgp = CMSG_FIRSTHDR(&recv_msg); cmsgp;
               cmsgp = CMSG_NXTHDR(&recv_msg, cmsgp)) {
               if (cmsgp->cmsg_level == SOL_SOCKET &&
                  cmsgp->cmsg_type == SCM_UCRED) {
                  ucred = (ucred_t *)CMSG_DATA(cmsgp);
                     break;
                     }

               if (ucred == NULL) {
                   (void) sprintf(&message[0],
                       "No ucred info in ancillary data with UDP");
               } else {
                   /*
                    * ここで、ucred_getlabel(3C) を使用して、ucred から
                    * ラベルを抽出することもできる。
                    */
               }

           }

           [...]

           if (message != NULL)
```

```
                (void) strlcpy(&inmsg[0], message, MAX_MSGLEN);
            /*
             * 正しいラベルが含まれるように、受信したメッセージを
             * 使用する
             */
            iov.iov_len = strlen(inmsg);
            ret = sendmsg(sockfd, &recv_msg, 0);
        }
}
```

◆ ◆ ◆ 第 6 章

トラステッド X ウィンドウシステム

この章では、Trusted Extensions X ウィンドウシステム API について説明します。また、トラステッド X ウィンドウシステムのセキュリティーポリシーおよび Solaris Trusted Extensions インタフェースを説明するために使用する簡単な Motif アプリケーションも含まれています。

Trusted Extensions API を Solaris OS で使用する方法の例は、Solaris ソースコードを参照してください。OpenSolaris の Web サイト (`http://jp.opensolaris.org/`) の左のナビゲーションバーにある「Source Browser」をクリックします。Source Browser を使用して Solaris のソースコードを検索します。

この章の内容は次のとおりです。

トラステッド X ウィンドウシステムの環境

Solaris Trusted Extensions が構成されているシステムは、共通デスクトップ環境 (Common Desktop Environment、CDE) の拡張バージョンである Solaris Trusted Extensions CDE を使用します。Solaris Trusted Extensions CDE は、Trusted Extensions X ウィンドウシステムを使用します。Trusted Extensions X ウィンドウシステムには、必須アクセス制御 (MAC)、任意アクセス制御 (DAC)、および特権の使用をサポートするためのプロトコル拡張が含まれます。

データ転送セッションは「多インスタンス化」されます。すなわち、異なる機密ラベルおよびユーザー ID でセッションがインスタンス化されます。多インスタンス化は、1 つの機密ラベルまたはユーザー ID の非特権クライアントのデータが、別の機

密ラベルまたはユーザー ID の別のクライアントに転送されないようにします。このような転送は、トラステッド X ウィンドウシステムの DAC ポリシー、および等位書き込みと下位読み取りの MAC ポリシーに違反する可能性があります。

Trusted Extensions X ウィンドウシステム API によって、セキュリティー関連属性情報を取得したり設定したりできます。さらに、この API によって、テキスト文字列出力にスタイルを適用するフォントリストおよびピクセル幅を使用してラベルを文字列に変換できます。たとえば、14 ポイント、ボールドの Helvetica フォントなどです。このようなインタフェースは、通常、Motif ウィジェット、Xt イントリンシクス、Xlib、および CDE インタフェースとともに作成される管理アプリケーションによって呼び出されます。

- セキュリティー関連情報の取得 – これらのインタフェースは、X プロトコル要求が行われる Xlib レベルで機能します。入力パラメータ値に関するデータを取得するには、Xlib インタフェースを使用します。
- 文字列へのラベルの変換 – これらのインタフェースは Motif レベルで機能します。入力パラメータは、ラベル、テキスト文字列出力の体裁を指定するフォントリスト、および適切なピクセル幅です。指定したスタイルとピクセル幅の複合文字列が返されます。

これらのルーチンの宣言については、79 ページの「Trusted Extensions X ウィンドウシステム API」を参照してください。

トラステッド X ウィンドウシステムのセキュリティー属性

トラステッド X ウィンドウシステムのインタフェースは、さまざまな X ウィンドウシステムオブジェクトのセキュリティー関連属性の情報を管理します。Motif のみによって GUI アプリケーションを作成することができます。Motif アプリケーションは、Xlib オブジェクトのセキュリティー属性の情報を処理する Motif ウィジェットを基礎にして、Xlib オブジェクト ID を取得するために XToolkit ルーチンを使用します。

トラステッド X ウィンドウシステムインタフェースによってセキュリティー属性の情報が取得される X ウィンドウシステムのオブジェクトは、ウィンドウ、プロパティー、X ウィンドウサーバー、およびクライアントと X ウィンドウサーバー間の接続です。Xlib には、ウィンドウ、プロパティー、表示、およびクライアント接続 ID を取得する呼び出しがあります。

ウィンドウは、ユーザーに対して出力を表示し、クライアントからの入力を受け入れます。

プロパティーは、プロパティー名によってアクセスされるデータの任意の集まりです。プロパティー名およびプロパティータイプは「アトム」によって参照できます。これは一意の 32 ビットの識別子であり、文字の名前文字列です。

ウィンドウ、プロパティー、およびクライアント接続のセキュリティー属性は、所有者 ID および機密ラベル情報から構成されます。これらの属性を収集するための構造体については、80 ページの「X11 のデータ型」を参照してください。セキュリティー属性情報を取得および設定するインタフェースについては、79 ページの「Trusted Extensions X ウィンドウシステム API」を参照してください。

トラステッド X ウィンドウシステムのセキュリティーポリシー

ウィンドウ、プロパティー、およびピックスマップオブジェクトには、ユーザー ID、クライアント ID、および機密ラベルがあります。グラフィックコンテキスト、フォント、およびカーソルには、クライアント ID のみがあります。クライアントと X ウィンドウサーバーとの接続には、ユーザー ID、X ウィンドウサーバー ID、および機密ラベルがあります。

「ユーザー *ID*」は、オブジェクトを作成したクライアントの ID です。「クライアント *ID*」は、オブジェクトを作成するクライアントが接続される接続番号に関連します。

DAC ポリシーでは、オブジェクトを所有するクライアントはそのオブジェクトで操作を実行しなければなりません。クライアントのユーザー ID がオブジェクトの ID と等しい場合、クライアントはオブジェクトを所有します。接続要求の場合、クライアントのユーザー ID が X ウィンドウサーバーワークステーションの所有者のアクセス制御リスト (ACL) にある必要があります。あるいは、クライアントがトラステッドパス属性を表明する必要があります。

MAC ポリシーは、ウィンドウおよびピックスマップに関しては等位書き込み、ネーミングウィンドウに関しては等位読み取りです。MAC ポリシーは、プロパティーに関しては下位読み取りです。機密ラベルは、作成元クライアントの機密ラベルに設定されます。次は、それぞれのアクションの MAC ポリシーです。

- 変更、作成、削除 – クライアントの機密ラベルは、オブジェクトの機密ラベルと同等である必要があります。
- 名前付け、読み取り、取得 – クライアントの機密ラベルは、オブジェクトの機密ラベルより優位である必要があります。
- 接続要求 – クライアントの機密ラベルは、X ウィンドウサーバーワークステーションの所有者のセッション認可上限のほうが優位である必要があります。あるいは、クライアントがトラステッドパス属性を表明する必要があります。

ウィンドウは、クライアント間で共有される情報を含むプロパティーを持つことができます。ウィンドウプロパティーは、アプリケーションが実行されている機密ラベルで作成されるので、プロパティーデータへのアクセスはそれの機密ラベルによって分離されます。クライアントは、プロパティーを作成したり、ウィンドウ上

のプロパティーにデータを格納したり、MAC および DAC の制限に従ってプロパティーからデータを取得したりできます。多インスタンス化されないプロパティーを指定するには、TrustedExtensionsPolicy ファイルを更新します。

TrustedExtensionsPolicy ファイルは、Xsun サーバーおよび Xorg サーバー用にサポートされます。

- SPARC: Xsun 用のファイルが /usr/openwin/server/etc にあります。
- x86: Xorg 用のファイルが /usr/X11/lib/X11/xserver にあります。

次の節でそれぞれのセキュリティーポリシーについて説明します。

- ルートウィンドウ
- クライアントウィンドウ
- 優先指定/リダイレクトウィンドウ
- キーボード、ポインタ、サーバー制御
- 選択マネージャー
- デフォルトのウィンドウリソース
- ウィンドウ間のデータの移動

ルートウィンドウ

ルートウィンドウは、ウィンドウ階層の最上位です。ルートウィンドウは、いずれのクライアントにも属さない公共オブジェクトですが、保護する必要があるデータを持ちます。ルートウィンドウの属性は ADMIN_LOW で保護されます。

クライアントウィンドウ

クライアントには、通常、ルートウィンドウから派生する少なくとも 1 つの最上位クライアントウィンドウと、その最上位ウィンドウ内で入れ子にされた追加ウィンドウがあります。クライアントの最上位ウィンドウから派生するすべてのウィンドウの機密ラベルは同じです。

優先指定/リダイレクトウィンドウ

メニューや特定のダイアログボックスなどの優先指定/リダイレクトウィンドウは、別のクライアントから離れた入力フォーカスを受け取ることはできません。そのため、入力フォーカスは誤った機密ラベルでファイルに入力を受け入れません。優先指定/リダイレクトウィンドウは、作成元クライアントによって所有され、ほかのクライアントによって別の機密ラベルでデータにアクセスするために使用できません。

キーボード、ポインタ、サーバー制御

クライアントは、キーボード、ポインタ、およびサーバーを制御するために MAC および DAC を必要とします。フォーカスをリセットするには、クライアントはフォーカスを所有するか、実効セットに `win_devices` 特権を必要とします。

ポインタを移動するには、クライアントはポインタ制御、および宛先ウィンドウに対する MAC と DAC が必要です。明示的なユーザーアクションを含むイベントに関して、X および Y 座標情報が取得されます。

選択マネージャー

選択マネージャーアプリケーションは、信頼できないウィンドウ間をデータが転送されるカット＆ペースト、ドラッグ＆ドロップなど、ユーザーレベルのウィンドウ間データ移動を仲介します。転送が試みられると、選択マネージャーは転送をキャプチャーし、制御ユーザーの承認を確認し、ユーザーに確認とラベル付け情報を要求します。ユーザーがデータ移動を試みると常に、選択マネージャーが自動的に表示されます。選択マネージャーが表示されるようにアプリケーションコードを更新する必要はありません。

管理者は、一部の転送タイプに対して自動確認を設定できます。その場合、選択マネージャーは表示されません。転送が MAC および DAC ポリシーに一致する場合、データ転送は完了します。ファイルマネージャーおよびウィンドウマネージャーは、それぞれの専用ドロップ領域の選択エージェントとしても動作します。多インスタンス化される選択ターゲットを指定するには、`/usr/openwin/server/etc/TrustedExtensionsPolicy` ファイルを参照してください。自動的に確認される選択ターゲットを決定するには、`/usr/dt/config/sel_config` ファイルを参照してください。

デフォルトのウィンドウリソース

クライアントによって作成されていないリソースは、`ADMIN_LOW` で保護されるデフォルトのリソースです。`ADMIN_LOW` で実行されるか、適切な特権を持って実行されるクライアントのみがデフォルトのリソースを変更できます。

ウィンドウリソースは次のとおりです。

- ルートウィンドウの属性 – すべてのクライアントが読み取りおよび作成アクセスできますが、書き込みまたは変更アクセスは特権クライアントのみができます。78 ページの「特権操作とトラステッド X ウィンドウシステム」を参照してください。
- デフォルトカーソル – クライアントは、プロトコル要求でデフォルトカーソルを自由に参照できます。

- 事前定義アトム – TrustedExtensionsPolicyファイルには、事前定義アトムの読み取り専用リストが含まれます。

ウィンドウ間のデータの移動

クライアントは、選択マネージャーと関係なくウィンドウ間でデータを移動するには、実効セットにwin_selection特権が必要です。77 ページの「選択マネージャー」を参照してください。

特権操作とトラステッドXウィンドウシステム

ユーザーが介在せずにウィンドウ、プロパティー、またはアトム名にアクセスするライブラリルーチンは、MACおよびDACを必要とします。フレームバッファーグラフィックコンテキスト、フォント、およびカーソルにアクセスするライブラリルーチンは任意アクセスを必要とし、さらに、特別なタスクに対する追加特権を必要とする場合があります。

オブジェクトへのアクセスを拒否された場合、クライアントは実効セットにwin_dac_read、win_dac_write、 win_mac_read、win_mac_writeの特権のうち1つ以上を必要とする場合があります。これらの特権を有効または無効にするには、TrustedExtensionsPolicyファイルを参照してください。

次に、それぞれのタスクを実行するのに必要な特権を示します。

- ウィンドウリソースの構成と破棄 – Xウィンドウサーバーによって永続的に保持されるウィンドウまたはプロパティーを構成または破棄するには、クライアントプロセスは実効セットにwin_config特権が必要です。スクリーンセーバーはこのようなリソースの例です。
- ウィンドウ入力デバイスの使用 – キーボードおよびポインタ制御を取得および設定する、または、ポインタボタンのマッピングおよびキーボードのマッピングを変更するには、クライアントプロセスは実効セットにwin_devices特権が必要です。
- ダイレクトグラフィックスアクセスの使用 – ダイレクトグラフィックスアクセス(DGA) Xプロトコル拡張を使用するには、クライアントプロセスは実効セットにwin_dga特権が必要です。
- ウィンドウラベルのダウングレード – ウィンドウ、ピックスマップ、またはプロパティーの機密ラベルを既存のラベルより優位ではない新しいラベルに変更するには、クライアントプロセスは実効セットにwin_downgrade_sl特権が必要です。
- ウィンドウラベルのアップグレード – ウィンドウ、ピックスマップ、またはプロパティーの機密ラベルを既存のラベルより優位な新しいラベルに変更するには、クライアントプロセスは実効セットにwin_upgrade_sl特権が必要です。

- ウィンドウのフォントパスの設定 – フォントパスを変更するには、クライアントプロセスは実効セットに `win_fontpath` 特権が必要です。

Trusted Extensions X ウィンドウシステム API

Trusted X11 API を使用するには、次のヘッダーファイルが必要です。

```
#include <X11/extensions/Xtsol.h>
```

Trusted X11 の例は、`-lXtsol` および `-ltsol` ライブラリオプションによってコンパイルされます。

X11 ラベルクリッピング API を使用するには、次のヘッダーファイルが必要です。

```
#include <Dt/label_clipping.h>
```

ラベルクリッピングの例は、`-lDtTsol` および `-ltsol` ライブラリオプションによってコンパイルされます。

次の節で、Trusted X11 インタフェースおよび X11 ラベルクリッピングインタフェースのデータ型および宣言を示します。

- X11 のデータ型
- 属性へのアクセス
- ウィンドウラベルへのアクセスと設定
- ウィンドウユーザー ID へのアクセスと設定
- ウィンドウプロパティーラベルへのアクセスと設定
- ウィンドウプロパティーユーザー ID へのアクセスと設定
- ワークステーション所有者 ID へのアクセスと設定
- X ウィンドウサーバーの認可上限と最下位ラベルの設定
- トラステッドパスウィンドウでの作業
- スクリーンストライプの高さへのアクセスと設定
- ウィンドウの多インスタンス化情報の設定
- X11 ラベルクリッピングインタフェースでの作業

X11 のデータ型

次のデータ型が X11/extensions/Xtsol.h で定義されていて、Trusted Extensions X ウィンドウシステム API に使用されます。

- **X11** のオブジェクト型 – ResourceType 定義は、処理されるリソースの型を示します。値は IsWindow、IsPixmap、または IsColormap です。

  ResourceType は、認可上限を表す型定義です。インタフェースは型 m_label_t の構造体をパラメータとして受け入れ、同じ型の構造体に認可上限を返します。

- **X11** のオブジェクト属性 – XTsolResAttributes 構造体には、次のリソース属性が含まれます。

```
typedef struct _XTsolResAttributes {
    CARD32     ouid;    /* 所有者ユーザー ID */
    CARD32     uid;     /* ウィンドウのユーザー ID */
    m_label_t  *sl;     /* 機密ラベル */
} XTsolResAttributes;
```

- **X11** のプロパティー属性 – XTsolPropAttributes 構造体には、次のプロパティー属性が含まれます。

```
typedef struct _XTsolPropAttributes {
    CARD32     uid;     /* プロパティーのユーザー ID */
    m_label_t  *sl;     /* 機密ラベル */
} XTsolPropAttributes;
```

- **X11** のクライアント属性 – XTsolClientAttributes 構造体には、次のクライアント属性が含まれます。

```
typedef struct _XTsolClientAttributes {
    int      trustflag; /* クライアントが信頼できるとマークされている場合は真 */
    uid_t    uid;       /* クライアントを起動した所有者ユーザー ID */
    gid_t    gid;       /* グループ ID */
    pid_t    pid;       /* プロセス ID */
    u_long   sessionid; /* セッション ID */
    au_id_t  auditid;   /* 監査 ID */
    u_long   iaddr;     /* クライアントを実行するホストのインターネットアドレス */
} XTsolClientAttributes;
```

属性へのアクセス

リソース、プロパティー、およびクライアント属性にアクセスするために、次のルーチンが使用されます。

```
Status XTSOLgetResAttributes(Display *display, XID object, ResourceType type,
XTSOLResAttributes *winattrp);
```
このルーチンは、*winattrp* のウィンドウ ID のリソース属性を返します。`XTSOLgetResAttributes(3XTSOL)` のマニュアルページを参照してください。

```
Status XTSOLgetPropAttributes(Display *display, Window window, Atom property,
XTSOLPropAttributes *propattrp);
```
このルーチンは、*propattrp* のウィンドウ ID によって決まるプロパティーのプロパティー属性を返します。`XTSOLgetPropAttributes(3XTSOL)` のマニュアルページを参照してください。

```
Status XTSOLgetClientAttributes(Display *display, XID windowid,
XTsolClientAttributes *clientattrp);
```
このルーチンは *clientattrp* のクライアント属性を返します。`XTSOLgetClientAttributes(3XTSOL)` のマニュアルページを参照してください。

ウィンドウラベルへのアクセスと設定

ウィンドウの機密ラベルを取得および設定するために、`XTSOLgetResLabel()` および `XTSOLsetResLabel()` ルーチンが使用されます。

```
Status XTSOLgetResLabel(Display *display, XID object, ResourceType type,
m_label_t *sl);
```
このルーチンはウィンドウの機密ラベルを取得します。`XTSOLgetResLabel(3XTSOL)` のマニュアルページを参照してください。

```
Status XTSOLsetResLabel(Display *display, XID object, ResourceType type,
m_label_t *sl);
```
このルーチンはウィンドウの機密ラベルを設定します。`XTSOLsetResLabel(3XTSOL)` のマニュアルページを参照してください。

ウィンドウユーザー ID へのアクセスと設定

ウィンドウのユーザー ID を取得および設定するために、`XTSOLgetResUID()` および `XTSOLsetResUID()` ルーチンが使用されます。

```
Status XTSOLgetResUID(Display *display, XID object, ResourceType type, uid_t
*uidp);
```
このルーチンはウィンドウのユーザー ID を取得します。`XTSOLgetResUID(3XTSOL)` のマニュアルページを参照してください。

```
Status XTSOLsetResUID(Display *display, XID object, ResourceType type, uid_t
*uidp);
```
このルーチンはウィンドウのユーザー ID を設定します。`XTSOLsetResUID(3XTSOL)` のマニュアルページを参照してください。

ウィンドウプロパティーラベルへのアクセスと設定

ウィンドウ ID によって決まるプロパティーの機密ラベルを取得および設定するために、XTSOLgetPropLabel() および XTSOLsetPropLabel() ルーチンが使用されます。

```
Status XTSOLgetPropLabel(Display *display, Window window, Atom property,
m_label_t *sl);
```

このルーチンは、ウィンドウ ID によって決まるプロパティーの機密ラベルを取得します。XTSOLgetPropLabel(3XTSOL) のマニュアルページを参照してください。

```
Status XTSOLsetPropLabel(Display *display, Window window, Atom property,
m_label_t *sl);
```

このルーチンは、ウィンドウ ID によって決まるプロパティーの機密ラベルを設定します。XTSOLsetPropLabel(3XTSOL) のマニュアルページを参照してください。

ウィンドウプロパティーユーザー ID へのアクセスと設定

ウィンドウ ID によって決まるプロパティーのユーザー ID を取得および設定するために、XTSOLgetPropUID() および XTSOLsetPropUID() ルーチンが使用されます。

```
Status XTSOLgetPropUID(Display *display, Window window, Atom property, uid_t
*uidp);
```

このルーチンは、ウィンドウ ID によって決まるプロパティーのユーザー ID を取得します。XTSOLgetPropUID(3XTSOL) のマニュアルページを参照してください。

```
Status XTSOLsetPropUID(Display *display, Window window, Atom property, uid_t
*uidp);
```

このルーチンは、ウィンドウ ID によって決まるプロパティーのユーザー ID を設定します。XTSOLsetPropUID(3XTSOL) のマニュアルページを参照してください。

ワークステーション所有者 ID へのアクセスと設定

ワークステーションサーバーの所有者のユーザー ID を取得および設定するために、XTSOLgetWorkstationOwner() および XTSOLsetWorkstationOwner() ルーチンが使用されます。

注 – XTSOLsetWorkstationOwner() ルーチンはウィンドウマネージャーによってのみ使用されます。

`Status XTSOLgetWorkstationOwner(Display *display, uid_t *uidp);`
このルーチンは、ワークステーションサーバーの所有者のユーザー ID を取得します。`XTSOLgetWorkstationOwner(3XTSOL)` のマニュアルページを参照してください。

`Status XTSOLsetWorkstationOwner(Display *display, uid_t *uidp);`
このルーチンは、ワークステーションサーバーの所有者のユーザー ID を設定します。`XTSOLsetWorkstationOwner(3XTSOL)` のマニュアルページを参照してください。

X ウィンドウサーバーの認可上限と最下位ラベルの設定

X ウィンドウサーバーにセッション高位認可上限およびセッション最下位ラベルを設定するために、`XTSOLsetSessionHI()` および `XTSOLsetSessionLO()` ルーチンを使用します。セッション高位認可上限は、ラベルビルダー GUI から選択でき、ユーザーの範囲内である必要があります。セッション最下位ラベルは、マルチレベルセッションの場合のユーザーの最下位ラベルと同じです。

注 – これらのインタフェースはウィンドウマネージャーによってのみ使用されます。

`Status XTSOLsetSessionHI(Display *display, m_label_t *sl);`
セッションの高位認可上限は、ログイン時のワークステーション所有者の認可上限から設定されます。セッションの高位認可上限は、所有者の認可上限およびマシンモニターのラベル範囲の上限のほうが優位です。これが変更されると、ウィンドウサーバーの認可上限より高位の機密ラベルで実行されるクライアントからの接続要求は、特権がなければ拒否されます。`XTSOLsetSessionHI(3XTSOL)` のマニュアルページを参照してください。

`Status XTSOLsetSessionLO(Display *display, m_label_t *sl);`
セッションの最下位ラベルは、ログイン時のワークステーション所有者の最下位ラベルから設定されます。セッションの最下位ラベルは、管理のために設定されるユーザーの最下位ラベルおよびマシンモニターのラベル範囲の下限より高位です。この設定を変更すると、ウィンドウサーバーの機密ラベルより下位の機密ラベルで実行されるクライアントからの接続要求は、特権がなければ拒否されます。`XTSOLsetSessionLO(3XTSOL)` のマニュアルページを参照してください。。

トラステッドパスウィンドウでの作業

指定したウィンドウをトラステッドパスウィンドウにするため、および、指定したウインドウがトラステッドパスウィンドウであるかどうかを検査するために、`XTSOLMakeTPWindow()` および `XTSOLIsWindowTrusted()` ルーチンが使用されます。

`Status XTSOLMakeTPWindow(Display *display, Window *w);`
このルーチンは、指定したウィンドウをトラステッドパスウィンドウにします。`XTSOLMakeTPWindow(3XTSOL)` のマニュアルページを参照してください。

`Bool XTSOLIsWindowTrusted(Display *display, Window *window);`
このルーチンは、指定したウィンドウがトラステッドパスウィンドウであるかどうかを検査します。`XTSOLIsWindowTrusted(3XTSOL)` のマニュアルページを参照してください。

スクリーンストライプの高さへのアクセスと設定

スクリーンストライプの高さを取得および設定するために、`XTSOLgetSSHeight()` および `XTSOLsetSSHeight()` ルーチンが使用されます。

注 – これらのインタフェースはウィンドウマネージャーによってのみ使用されます。

`Status XTSOLgetSSHeight(Display *display, int screen_num, int *newHeight);`
このルーチンはスクリーンストライプの高さを取得します。`XTSOLgetSSHeight(3XTSOL)` のマニュアルページを参照してください。

`Status XTSOLsetSSHeight(Display *display, int screen_num, int newHeight);`
このルーチンはスクリーンストライプの高さを設定します。スクリーンストライプがなくなったり、非常に大きなスクリーンストライプになったりしないように注意してください。`XTSOLsetSSHeight(3XTSOL)` のマニュアルページを参照してください。

ウィンドウの多インスタンス化情報の設定

`Status XTSOLsetPolyInstInfo(Display *display, m_label_t sl, uid_t *uidp, int enabled);`
`XTSOLsetPolyInstInfo()` ルーチンは、クライアントとは異なる機密ラベルで、クライアントがプロパティーからプロパティー情報を取得できるようにします。最初の呼び出しで、必要な機密ラベルとユーザー ID を指定し、`enabled` プロパティーを True に設定します。次に、`XTSOLgetPropAttributes()`、`XTSOLgetPropLabel()`、または `XTSOLgetPropUID()` を呼び出します。終了するには、`enabled` プロパティーを False に設定して `XTSOLsetPolyInstInfo()` ルーチンを再び呼び出します。`XTSOLsetPolyInstInfo(3XTSOL)` のマニュアルページを参照してください。

X11ラベルクリッピングインタフェースでの作業

```
int label_to_str(const m_label_t *label, char **string, const m_label_str_t
conversion_type, uint_t flags);
```

`label_to_str()`ルーチンは、機密ラベルまたは認可上限を文字列に変換します。`label_to_str(3TSOL)`のマニュアルページを参照してください。

トラステッドXウィンドウシステムインタフェースの使用

このあとの各節では、Trusted Extensionsインタフェース呼び出しを使用するコーディング例の抜粋を示します。これらの呼び出しはセキュリティー属性を処理して、ラベルを文字列に変換します。抜粋では、ウィンドウセキュリティー属性、アプリケーションプログラムでもっとも一般的に管理される属性の処理に焦点を当てます。多くの場合、クライアントは、別のアプリケーションによって作成されたオブジェクトに対して適切な特権を使用することによってセキュリティー属性を取得します。その次に、クライアントは属性をチェックして、オブジェクトに対する操作がシステムのセキュリティーポリシーによって許可されているか否かを判別します。このセキュリティーポリシーには、DACポリシー、およびMACの等位書き込みと下位読み取りのポリシーが含まれます。アクセスが拒否されると、アプリケーションは場合に応じてエラーを生成するか、特権を使用します。特権が必要とされる場合の説明は、78ページの「特権操作とトラステッドXウィンドウシステム」を参照してください。

Trusted Extensions APIに渡すためのオブジェクトのIDを取得する前に、オブジェクトを作成してください。

ウィンドウ属性の取得

`XTSOLgetResAttributes()`ルーチンは、ウィンドウのセキュリティー関連属性を返します。次を指定します。

- 表示ID
- ウィンドウID
- セキュリティー属性を必要とするオブジェクトがウィンドウであることを示すフラグ
- 返される属性を取得する`XtsolResAttributes`構造体

クライアントは、そのクライアントが作成したウィンドウのセキュリティー属性を取得するので、特権は必要ありません。

このマニュアル内のプログラム例は、APIの紹介を中心としていて、エラーチェックは行なっていません。実際に作成するアプリケーションでは、適切なエラーチェックを実行してください。

```
/* Xlib 呼び出しによって基本のウィンドウと表示 ID を取得する */
   window = XtWindow(topLevel);
   display = XtDisplay(topLevel);

/* ウィンドウセキュリティー属性を取得する */
   retval = XTSOLgetResAttributes(display, window, IsWindow, &winattrs);

/* ラベルを文字列に変換する */
   retval = label_to_str(&winattrs.sl, &plabel, M_LABEL, LONG_NAMES);

/* セキュリティー属性情報を出力する */
   printf("Workstation Owner ID = %d\nUser ID = %d\nLabel = %s\n",
   winattrs.ouid, winattrs.uid, string1);
```

printf 文によって次のように出力されます。

```
Workstation Owner ID = 29378
User ID = 29378
Label = CONFIDENTIAL
```

フォントリストによるウィンドウラベルの変換

この例は、プロセス機密ラベルを取得し、フォントリストおよびピクセル幅を使用してそれを文字列に変換する方法を示します。ラベルの文字列とともに、ラベルウィジェットが作成されます。プロセス機密ラベルはウィンドウ機密ラベルと同じです。したがって、特権は必要ありません。

最後の文字列がピクセル幅に収まらない場合、文字列は切り取られて切り取りインジケータが表示されます。X ウィンドウシステムのラベル変換インタフェースは指定したピクセル数で切り取り、ラベルクリッピングインタフェースは文字数で切り取ります。

注 – 英語以外の言語で label_encodings ファイルを使用する場合、ISO 標準のコード番号 128 以上のアクセント文字では変換されないことがあります。次の例は、アジア言語の文字セットでは機能しません。

```
    retval = getplabel(&senslabel);

/* フォントリストを作成し、それを使用してラベルを変換する */
    italic = XLoadQueryFont(XtDisplay(topLevel),
         "-adobe-times-medium-i-*-*-14-*-*-*-*-*-iso8859-1");
    fontlist = XmFontListCreate(italic, "italic");
    xmstr = Xbsltos(XtDisplay(topLevel), &senslabel, width, fontlist,
        LONG_WORDS);
```

```
/* フォントリストとラベルテキストを使用してラベルウィジェットを作成する */
    i=0;
    XtSetArg(args[i], XmNfontList, fontlist); i++;
    XtSetArg(args[i], XmNlabelString, xmstr); i++;
    label = XtCreateManagedWidget(“label”, xmLabelWidgetClass,
        form, args, i);
```

ウィンドウラベルの取得

この例は、ウィンドウの機密ラベルを取得する方法を示します。プロセス機密ラベルはウィンドウ機密ラベルと同じです。したがって、特権は必要ありません。

```
/* ウィンドウラベルを取得する */
    retval = XTSOLgetResLabel(display, window, IsWindow, &senslabel);

/* ラベルを文字列に変換し、出力する */
    retval = label_to_str(label, &string, M_LABEL, LONG_NAMES);
    printf(“Label = %s\n”, string);
```

`printf` 文によって、たとえば、次のように出力されます。

```
Label = PUBLIC
```

ウィンドウラベルの設定

この例は、ウィンドウに機密ラベルを設定する方法を示します。新しい機密ラベルは、ウィンドウおよびプロセスの機密ラベルより優位です。クライアントが優位ではないラベルを変換するには、クライアントの実効セットに `sys_trans_label` 特権が必要です。さらに、ウィンドウの機密ラベルを変更するには、クライアントに `win_upgrade_sl` 特権も必要です。

特権の使用についての詳細は、『Solaris セキュリティーサービス開発ガイド』を参照してください。

```
/* テキスト文字列を機密ラベルに変換する */
   retval = label_to_str(string4, &label, M_LABEL, L_NO_CORRECTION, &error);

/* 新しい値で機密ラベルを設定する */
   retval = XTSOLsetResLabel(display, window, IsWindow, label);
```

ウィンドウユーザー ID の取得

この例は、ウィンドウユーザー ID を取得する方法を示します。プロセスはウィンドウリソースを所有し、同じ機密ラベルで実行されています。したがって、特権は必要ありません。

```
/* ウィンドウのユーザー ID を取得する */
    retval = XTSOLgetResUID(display, window, IsWindow, &uid);
```

X ウィンドウサーバーワークステーション所有者 ID の取得

この例は、X ウィンドウサーバーにログインしているユーザーの ID を取得する方法を示します。プロセス機密ラベルはウィンドウ機密ラベルと同じです。したがって、特権は必要ありません。

```
/* ウィンドウのユーザー ID を取得する */
    retval = XTSOLgetWorkstationOwner(display, &uid);
```

◆ ◆ ◆ 第 7 章

7 ラベルビルダー API

Solaris Trusted Extensions は、Motif ベースの API のセットを提供します。これらのインタフェースを使用することによって、ユーザー入力から有効な機密ラベルまたは認可上限を作成するための対話型 GUI を作成できます。これらのインタフェースは「ラベルビルダー *API*」と呼ばれます。多くの場合、これらの API は管理アプリケーションから呼び出されます。

ラベルビルダー GUI は Solaris Trusted Extensions が構成されているシステム で使用されます。『Solaris Trusted Extensions ユーザーズガイド』では、これらのインタフェースについてエンドユーザーの観点から説明しており、ラベルビルダーライブラリルーチンによって提供される機能も示されています。

Trusted Extensions API を Solaris OS で使用する方法の例は、Solaris ソースコードを参照してください。OpenSolaris の Web サイト (`http://jp.opensolaris.org/`) の左のナビゲーションバーにある「Source Browser」をクリックします。Source Browser を使用して Solaris のソースコードを検索します。

この章の内容は次のとおりです。

ラベルビルダー GUI 用の API

この節で説明する API を使用するには、次のヘッダーファイルを組み込む必要があります。

```
#include <Dt/ModLabel.h>
```

ライブラリビルダーの例は、`-lDtTsol` および `-ltsol` ライブラリオプションによってコンパイルされます。

次の API は、ラベル GUI の作成のために使用できます。データ型とパラメータのリストは90 ページの「対話型ユーザーインタフェースの作成」にあります。

```
ModLabelData *tsol_lbuild_create(Widget widget, void (*event_handler)()
ok_callback, lbuild_attributes extended_operation, ..., NULL);
```

`tsol_lbuild_create()` ルーチンは GUI を作成し、ユーザーインタフェースに関する情報を含む型 `ModLabeldata` のポインタ変数を返します。この情報は、`tsol_lbuild_create()` 入力パラメータリストに渡される値、指定されない情報に代わるデフォルト値、およびユーザーインタフェースの作成にラベルビルダーが使用するウィジェットに関する情報の組み合わせです。

`LBUILD_WORK_SL` と `LBUILD_WORK_CLR` の操作値は、ユーザーが指定する入力によって設定される値なので、`tsol_lbuild_create()` には有効ではありません。

`tsol_lbuild_get()` および `tsol_lbuild_set()` ルーチンを使用して、拡張操作と値を取得および設定できます。ただし、これらのルーチンはウィジェット情報には使用できません。この情報は、`ModLabelData` 構造体のフィールドを参照することによって直接アクセスされます。`labelbuilder(3TSOL)` のマニュアルページを参照してください。

```
void tsol_lbuild_destroy(ModLabelData *lbdata);
```

`tsol_lbuild_destroy()` ルーチンは、`tsol_lbuild_create()` ルーチンによって返される `ModLabelData` 構造体を破棄します。

```
void *tsol_lbuild_get(ModLabelData *lbdata, lbuild_attributes
extended_operation);
```

`tsol_lbuild_get()` ルーチンは、`tsol_lbuild_create()` によって作成されて `ModLabelData` 構造体に格納されるユーザーインタフェース情報にアクセスします。

```
void tsol_lbuild_set(ModLabelData *lbdata, lbuild_attributes
extended_operation, ..., NULL);
```

`tsol_lbuild_set()` ルーチンは、`tsol_lbuild_create()` によって作成されて `ModLabelData` 構造体に格納されるユーザーインタフェース情報を変更します。`LBUILD_WORK_SL` と `LBUILD_WORK_CLR` の操作値は、ユーザーが指定する入力によって設定される値なので、`tsol_lbuild_set()` には有効ではありません。

対話型ユーザーインタフェースの作成

次の図は、図のあとに示すコードによって作成される GUI です。`main` プログラムは、1 つのボタン (*display*) を持つ親フォーム (*form*) を作成します。ボタンコールバックが、`tsol_lbuild_create()` ルーチンの呼び出しによって作成されるラベルビルダーのダイアログボックスを表示します。`tsol_lbuild_create(3TSOL)` のマニュアルページを参照してください。

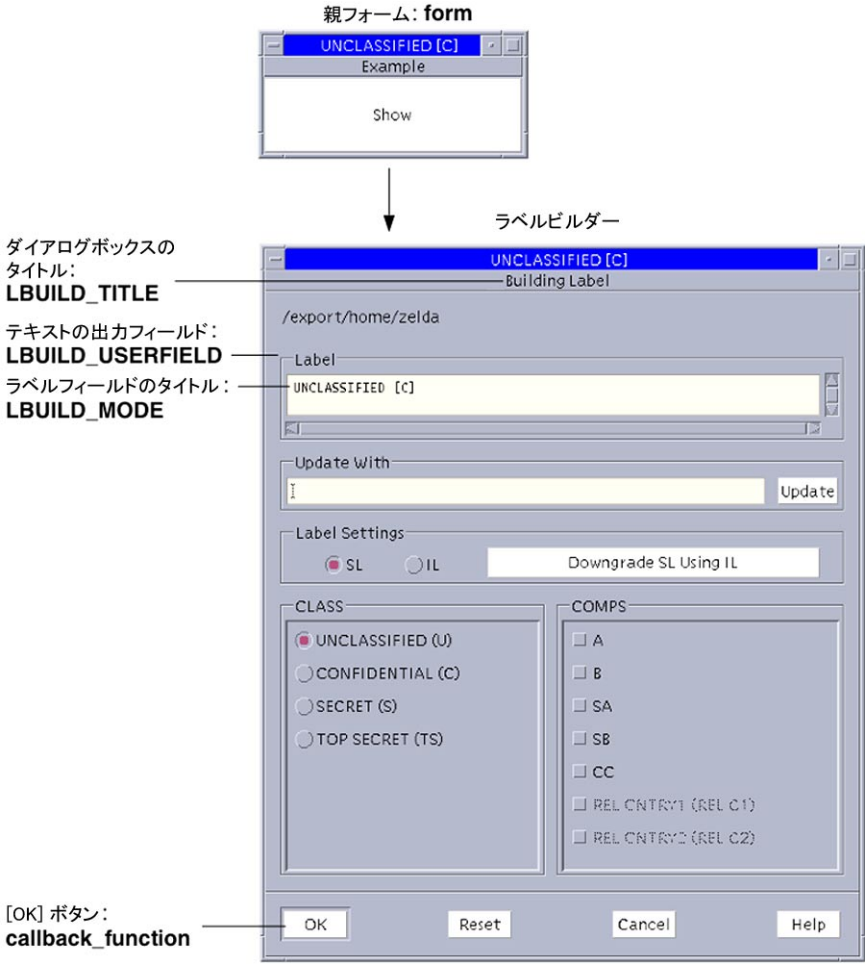

図7–1 ラベル作成のインタフェース

親フォームの「Show」ボタンをクリックすると、ラベルビルダーのダイアログボックスが表示されます。`tsol_lbuild_create()` ルーチンに渡されるパラメータがラベルビルダーのダイアログボックスのどこにあるかが引出線によって示されています。`tsol_lbuild_create(3TSOL)` のマニュアルページを参照してください。

次のコードによって、図に示すような GUI が作成されます。

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/PushB.h>
#include <Xm/Form.h>
```

```
#include <Dt/ModLabel.h>

ModLabelData *data;

/* tsol_lbuild_create() に渡されるコールバック */
void callback_function()
{
    char *title, *userval;
    char *string = (char *)0;
    char *string1 = (char *)0;
    int mode, view;
    Boolean show;
    m_label_t *sl_label, *work_sl_label;
    Position x, y;

/* アプリケーション固有の実装を行う */
    printf("OK button called\n");

/* 照会の設定 */
    mode = (int)tsol_lbuild_get(data, LBUILD_MODE);
    title = (String)tsol_lbuild_get(data, LBUILD_TITLE);
    sl_label = (m_label_t*) tsol_lbuild_get(data, LBUILD_VALUE_SL);
    work_sl_label = (m_label_t*) tsol_lbuild_get(data, LBUILD_WORK_SL);
    view = (int)tsol_lbuild_get(data, LBUILD_VIEW);
    x = (Position ) tsol_lbuild_get(data, LBUILD_X);
    y = (Position ) tsol_lbuild_get(data, LBUILD_Y);
    userval = (char *)tsol_lbuild_get(data, LBUILD_USERFIELD);
    show = (Boolean )tsol_lbuild_get(data, LBUILD_SHOW);

    label_to_str(sl_label, &string, M_LABEL, LONG_NAMES);
    label_to_str(work_sl_label, &string1, M_LABEL, LONG_NAMES);
    printf("Mode = %d, Title = %s, SL = %s, WorkSL = %s, View = %d, ",
        mode, title, string, string1, view);
    printf("X = %d, Y = %d, Userval = %s, Show = %d\n",
        x, y, userval, show);

}

/* ボタンが押されたらダイアログボックスを表示するコールバック */
void Show(Widget display, caddr_t client_data, caddr_t call_data)
{
    tsol_lbuild_set(data, LBUILD_SHOW, TRUE, NULL);
}

main(int argc, char **argv)
{
    Widget     form, topLevel, display;
    Arg args[9];
```

```
    int i = 0, error, retval;
    char *sl_string = "CNF";
    m_label_t * sl_label;

    topLevel = XtInitialize(argv[0], "XMcmds1", NULL, 0, &argc, argv);
    form = XtCreateManagedWidget("form",
        xmFormWidgetClass, topLevel, NULL, 0);

    retval = str_to_label(sl_string, &sl_label, MAC_LABEL, L_NO_CORRECTION, NULL);
    printf("Retval = %d\n", retval);

    data = tsol_lbuild_create( form, callback_function,
        LBUILD_MODE, LBUILD_MODE_SL,
        LBUILD_TITLE, "Building Sensitivity Label",
        LBUILD_VALUE_SL, sl_label,
        LBUILD_VIEW, LBUILD_VIEW_EXTERNAL,
        LBUILD_X, 200,
        LBUILD_Y, 200,
        LBUILD_USERFIELD, "/export/home/zelda",
        LBUILD_SHOW, FALSE,
        NULL);

    i = 0;
    XtSetArg(args[i], XmNtopAttachment, XmATTACH_FORM); i++;
    XtSetArg(args[i], XmNleftAttachment, XmATTACH_FORM); i++;
    XtSetArg(args[i], XmNrightAttachment, XmATTACH_FORM); i++;
    XtSetArg(args[i], XmNbottomAttachment, XmATTACH_FORM); i++;
    display = XtCreateManagedWidget("Show",
        xmPushButtonWidgetClass, form, args, i);
    XtAddCallback(display, XmNactivateCallback, Show,0);
    XtRealizeWidget(topLevel);

    XtMainLoop();

    tsol_lbuild_destroy(data);

}
```

プログラムを実行すると、次の出力が生成されます。

```
OK button called
Mode = 12, Title = Building Sensitivity label,
Label = CNF, WorkSL = SECRET,
View = 1, X = 200, Y = 200,
Userval = /export/home/zelda,
Show = 1
```

この節の内容は次のとおりです。

- ラベルビルダーの動作
- ラベルビルダーのアプリケーション固有の機能
- 特権操作とラベルビルダー
- `tsol_lbuild_create()` ルーチン
- 拡張ラベルビルダー操作
- `ModLabelData` 構造体

ラベルビルダーの動作

ラベルビルダーのダイアログボックスは、エンドユーザーに情報の入力を求め、その入力から有効な機密ラベルを生成します。ラベルビルダーは、有効なラベルまたは認可上限が確実に作成されるようにします。ラベルおよび認可上限は、システムの `label_encodings` ファイルに定義されます。

ラベルビルダーは、「OK」、「Reset」、「Cancel」、「Update」ボタンに対するデフォルト動作を提供します。`tsol_lbuild_create()` ルーチンに渡されるコールバックは、「OK」ボタンにマップされてアプリケーション固有の動作を提供します。

キーボード入力と「Update」ボタン

「Update」ボタンは、「Update With」フィールドにユーザーが入力するテキストを受け取り、その文字列が `label_encodings` ファイルに定義されている有効なラベルまたは認可上限であることを検査します。

- 入力が有効でない場合、ラベルビルダーはユーザーにエラーを表示します。
- 入力が有効である場合、ラベルビルダーは「Label」フィールドのテキストを更新し、`tsol_lbuild_create()` ルーチンによって返される `ModLabelData` 変数の該当する作業ラベルフィールドにその値を格納します。98 ページの「`ModLabelData` 構造体」を参照してください。

ユーザーが「OK」をクリックすると、「OK」ボタンのコールバック実装に従ってユーザー作成の値が処理されます。

ラジオボタンオプション

「Label Settings」ラジオボタンオプションによって、格付けおよびコンパートメントから機密ラベルまたは認可上限を作成できます。また、このオプションでは、格付け、コンパートメント、およびマーキングから情報ラベルも作成できます。モードによっては、ボタンのいずれかがグレー表示になることがあります。この方法は、キーボード入力による方法、および前の節で説明した「Update」ボタンによる方法とは別です。

格付け、コンパートメント、およびマーキングに関する情報は、システムの label_encodings ファイルに指定します。label_encodings ファイルに指定される組み合わせおよび制約が強制されますが、無効な組み合わせはグレー表示です。ユーザーがオプションを選択すると、「Label」フィールドが更新され、tsol_lbuild_create() ルーチンによって返される ModLabelData 変数の該当する作業ラベルフィールドにその値が格納されます。ユーザーは格付け (CLASS) およびコンパートメント (COMPS) のリストにあるラジオボタンを選択することによって、機密ラベルまたは認可上限を作成できます。

ユーザーが「OK」をクリックすると、「OK」ボタンのコールバック実装に従ってユーザー作成の値が処理されます。

「Reset」ボタン

「Reset」ボタンは、「Label」フィールドのテキストをアプリケーションが起動したときの値に設定します。

「Cancel」ボタン

「Cancel」ボタンは、変更内容を保存せずにアプリケーションを終了します。

ラベルビルダーのアプリケーション固有の機能

ラベルビルダー GUI は、有効なラベルまたは認可上限を生成します。アプリケーション固有のコールバック、エラー処理、さらにラベルまたは認可上限に関連付けられるその他の機能も追加する必要があります。

特権操作とラベルビルダー

ラベルビルダーは、ワークスペース機密ラベルが優位である格付けおよび関連コンパートメントのみをユーザーに表示します。実行可能ファイルの実効セットに sys_trans_label 特権がある場合、それ以外の格付けおよびコンパートメントも表示されることがあります。

「OK」ボタンコールバックのアプリケーション固有の実装には、特権が必要な場合があります。

ユーザーがラベルをアップグレードまたはダウングレードする承認を持っていない場合、「OK」および「Reset」ボタンはグレー表示です。ユーザー作成のラベルがユーザーの範囲外の場合も同様です。グレー表示のボタンでは、ユーザーはタスクを完了できません。この制限を無効にできる特権はありません。

tsol_lbuild_create() ルーチン

tsol_lbuild_create() ルーチンは、ウィジェットのいずれも、コールバック関数、および NULL で終わる一連の名前と値のペアを受け入れます。名前が操作を表します。ルーチンは型 ModLabelData の変数を返します。

次に、tsol_lbuild_create() ルーチンによって受け入れられる情報について説明します。

- ウィジェット – ラベルビルダーは、いずれのウィジェットからもダイアログボックスを作成できます。
- コールバック関数 – 「OK」ボタンがクリックされると、コールバック関数が起動されます。このコールバック関数はアプリケーション固有の動作を提供します。
- 名前と値のペア – ペアの名前 (左) 側は拡張操作を指定し (96 ページの「拡張ラベルビルダー操作」を参照)、値 (右) 側はその値を指定します。値は列挙型定数である場合があります。それ以外の場合は、値を指定します。ペアの指定順序は任意ですが、指定するすべての操作に有効な値が必要です。

  戻り値は、作成したダイアログボックスに関する情報を含むデータ構造体です。情報の元は、tsol_lbuild_create() 入力パラメータおよび実行時のユーザーアクティビティーです。ラベルビルダーは、値が指定されていない一部のフィールドにデフォルト値を提供します。

  プログラムによってこの名前と値のペアの情報にアクセスして変更するには、tsol_lbuild_get() ルーチンおよび tsol_lbuild_set() ルーチンを使用します。データ構造については、98 ページの「ModLabelData 構造体」を参照してください。

  次の例は、tsol_lbuild_create() ルーチンの呼び出しです。

```
data= tsol_lbuild_create(form, callback_function,
    LBUILD_MODE, LBUILD_MODE_SL,
    LBUILD_TITLE, "Building a Label",
    LBUILD_VALUE_SL, sl_label,
    LBUILD_VIEW, LBUILD_VIEW_EXTERNAL,
    LBUILD_X, 200,
    LBUILD_Y, 200,
    LBUILD_USERFIELD “/export/home/zelda”,
    LBUILD_SHOW, FALSE,
    NULL);
```

拡張ラベルビルダー操作

この節では、拡張操作と、tsol_lbuild_create()、tsol_lbuild_get()、および tsol_lbuild_set() ルーチンに渡すことができる有効な値について説明します。tsol_lbuild_create() に渡される値は、それの戻り値に格納されます。戻り値の型は

ModLabelData です。パラメータに返される値には、tsol_lbuild_get() および tsol_lbuild_set() の呼び出しによってアクセスできます。ModLabelData 構造体は、98 ページの「ModLabelData 構造体」で説明されています。
tsol_lbuild_create(3TSOL)、tsol_lbuild_get(3TSOL)、および tsol_lbuild_set(3TSOL) のマニュアルページを参照してください。

すべての拡張操作は、tsol_lbuild_get() に渡すのに有効です。ただし、LBUILD_WORK_SL および LBUILD_WORK_CLR の操作は、tsol_lbuild_set() または tsol_lbuild_create() に渡すのに有効ではありません。これらの値はユーザー入力に基づいてラベルビルダーによって設定されるためです。この例外は、次の操作の説明にも記されています。

- LBUILD_MODE – 機密ラベルまたは認可上限を作成するためのユーザーインタフェースを作成するように tsol_lbuild_create() に指示することができます。デフォルト値は LBUILD_MODE_SL です。
  - LBUILD_MODE_SL – 機密ラベルを作成します。
  - LBUILD_MODE_CLR – 認可上限を作成します。
- LBUILD_VALUE_SL – モードが LBUILD_MODE_SL のときに「Label」フィールドに表示される最初の機密ラベル。デフォルト値は ADMIN_LOW です。
- LBUILD_VALUE_CLR – モードが LBUILD_MODE_CLR のときに「Label」フィールドに表示される最初の認可上限。デフォルト値は ADMIN_LOW です。
- LBUILD_USERFIELD – ラベルビルダーのダイアログボックスの最上部に表示される文字列プロンプト。 デフォルト値は NULL です。
- LBUILD_SHOW – ラベルビルダーのダイアログボックスを表示または非表示にします。デフォルト値は FALSE です。
  - TRUE – ラベルビルダーのダイアログボックスを表示します。
  - FALSE – ラベルビルダーのダイアログボックスを非表示にします。
- LBUILD_TITLE – ラベルビルダーのダイアログボックスの最上部に表示される文字列タイトル。デフォルト値は NULL です。
- LBUILD_WORK_SL – ユーザーが作成する機密ラベル。ユーザーが「Update」ボタンを選択したり、対話式にオプションを選択すると、この値はユーザーの入力に基づいて更新されます。デフォルト値は ADMIN_LOW です。tsol_lbuild_set() または tsol_lbuild_create() に対して有効な拡張操作ではありません。
- LBUILD_WORK_CLR – ユーザーが作成する認可上限。ユーザーが「Update」ボタンを選択したり、対話式にオプションを選択すると、この値はユーザーの入力に基づいて更新されます。デフォルト値は ADMIN_LOW です。tsol_lbuild_set() または tsol_lbuild_create() に対して有効な拡張操作ではありません。
- LBUILD_X – 画面の左上隅に対する、ラベルビルダーのダイアログボックスの左上隅からの X 軸方向のオフセット (ピクセル単位)。デフォルトでは、ラベルビルダーのダイアログボックスは画面中央に配置されます。

- LBUILD_Y – 画面の左上隅に対する、ラベルビルダーのダイアログボックスの左上隅からの Y 軸方向のオフセット (ピクセル単位)。デフォルトでは、ラベルビルダーのダイアログボックスは画面中央に配置されます。
- LBUILD_UPPER_BOUND – ラジオボタンとしてユーザーに使用可能な最高位格付け、および関連コンパートメントとマーキング。これらのボタンは、ラベルまたは認可上限を対話式に作成するために使用されます。指定する値はユーザーの範囲内である必要があります。値を指定しない場合、この値はユーザーのワークスペース機密ラベルになります。または、実行可能ファイルが sys_trans_label 特権を持つ場合、この値はユーザーの認可上限です。
- LBUILD_LOWER_BOUND – ラジオボタンとしてユーザーに使用可能な最下位格付け、および関連コンパートメントとマーキング。これらのボタンは、ラベルまたは認可上限を対話式に作成するために使用されます。この値はユーザーの最小ラベルです。値を指定しない場合、この値はユーザーの属性によって指定されるデフォルトに基づきます。
- LBUILD_CHECK_AR – ユーザー作成のラベルがユーザーの範囲内であるかを検査します。値 1 は「検査する」、値 0 は「検査しない」です。ラベルが範囲外の場合、エラーメッセージがユーザーに表示されます。デフォルト値は 1 です。
- LBUILD_VIEW – 内部ラベル表現または外部ラベル表現を使用するか否かを決定します。デフォルト値は LBUILD_VIEW_EXTERNAL です。
  - LBUILD_VIEW_INTERNAL – システムの最上位ラベルおよび最下位ラベルの内部名 (ADMIN_HIGH および ADMIN_LOW) を使用します。
  - LBUILD_VIEW_EXTERNAL – ADMIN_LOW ラベルを次の最下位ラベルに上げ、ADMIN_HIGH ラベルを次の最上位ラベルに下げます。

ModLabelData 構造体

ModLabelData 構造体には、tsol_lbuild_create() ルーチンを呼び出すことによって作成されるラベルビルダーインタフェースの状態に関する情報が含まれます。次の表に ModLabelData フィールドを示します。ウィジェットとコールバックを除くすべてのフィールドには、関連付けられている拡張操作を指定することによって、また tsol_lbuild_set() または tsol_lbuild_get() の呼び出しにおいて有効な値を指定することによってアクセスできます。拡張操作の説明は、96 ページの「拡張ラベルビルダー操作」を参照してください。

表 7–1 ModLabelData 構造体

拡張操作、または摘要	データ型	フィールド	コメント
LBUILD_CHECK_AR	int	check_ar	
LBUILD_MODE	int	mode	

表 7–1 ModLabelData 構造体 *(続き)*

拡張操作、または摘要	データ型	フィールド	コメント
LBUILD_SHOW	Bool	show	
LBUILD_TITLE	char	*lbuild_title	
LBUILD_UPPER_BOUND, LBUILD_LOWER_BOUND	brange_t	range	
LBUILD_USERFIELD	char	*userfield	
LBUILD_VALUE_CLR	bclear_t	*clr	
LBUILD_VALUE_SL	m_label_t	*sl	
LBUILD_VIEW	int	view	
LBUILD_WORK_CLR	bclear_t	*clr_work	tsol_lbuild_set() または tsol_lbuild_create() には無効
LBUILD_WORK_SL	m_label_t	*sl_work	tsol_lbuild_set() または tsol_lbuild_create() には無効
LBUILD_X	Position	x	
LBUILD_Y	Position	y	
tsol_lbuild_create() に渡されるコールバック	void	(*event_handler)()	
「Cancel」ボタン	Widget	cancel	
「Help」ボタン	Widget	help	
ラベルビルダーのダイアログボックス	Widget	lbuild_dialog	
「OK」ボタン	Widget	ok	
「Reset」ボタン	Widget	reset	
「Update」ボタン	Widget	update	

ラベルビルダーのオンラインヘルプ

「Help」ボタンと、ユーザーインタフェースに使用されるその他のウィジェットは、ModLabelData 構造体の lbl_shell フィールドを通じてアプリケーションコードから直接アクセスできます。アプリケーションにオンラインヘルプを追加するには、『共通デスクトップ環境 プログラマーズ・ガイド (ヘルプ・システム編)』の手順とガイドラインに従います。

◆ ◆ ◆ 第 8 章

信頼できるWebガードプロトタイプ

この章では、Webガードと呼ばれる安全なWebブラウジングプロトタイプの構成について説明します。Webガードは、Webサーバーとその Webコンテンツを隔離するように構成して、インターネットからの攻撃を防止します。

この章で説明するWebガードプロトタイプは、完全なソリューションではありません。これは、マルチレベルポートがラベルの境界をまたがるプロキシ URL要求のためにどのように使用できるかを示すプロトタイプです。より完全なソリューションにするには、承認、データフィルタリング、監査などを組み込みます。

プロトタイプで主に実装されるのは管理機能です。プロトタイプは、マルチレベルポート、トラステッドネットワーク、およびApache Webサーバー構成を使用してWebガードを設定します。管理を目的にする例のほかに、プログラム上の方法を使用して安全なWebブラウジングプロトタイプを設定できます。

この章の内容は次のとおりです。

管理のための Web ガードプロトタイプ

この節では、Webサーバーとその Webコンテンツを隔離するように構成して、インターネットからの攻撃を防止する安全なWebブラウジングプロトタイプの例を示します。このWebガードプロトタイプは、管理上のトラステッドネットワーク機能を使用して2段階のフィルタを構成することによって、保護されているWebサーバーとWebコンテンツへのアクセスを制限します。このプロトタイプは、管理上の方法のみによって実装されています。プログラミングの必要はありません。

次の図は、マルチレベル環境のWebガードプロトタイプの構成を示します。ラベルが図でどのように配置されるかによって、ラベル関係が示されます。縦の関係はラベルの上下を表し、横の関係は分離ラベルを表します。

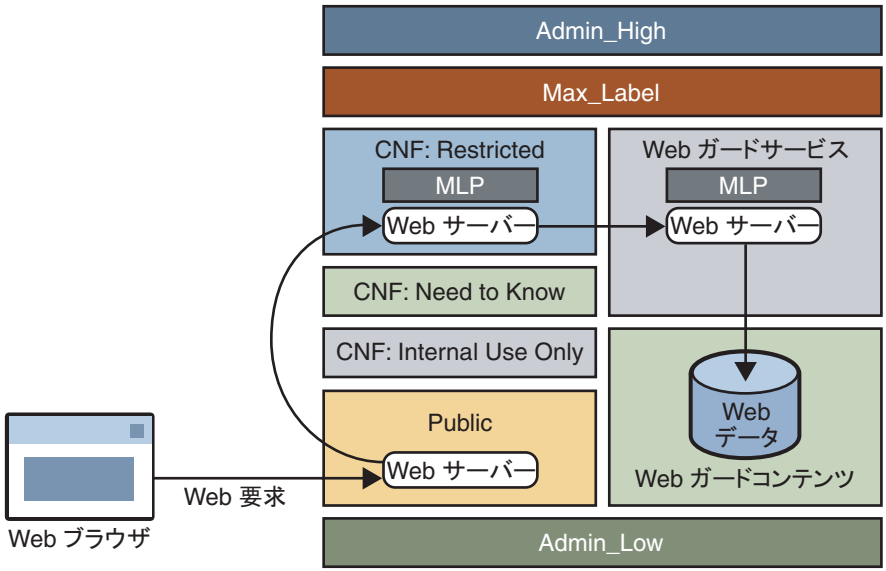

図 8–1 Web ガードの構成

Web 要求は、public ゾーンで構成されている Web サーバーに入り、restricted ゾーンで構成されている Web サーバーに渡されます。

restricted ゾーンは、マルチレベルポート (MLP) を使用して public ゾーンのポート 8080 で要求を待機します。この Web サーバーは要求を webservice のラベル付けされたゾーンに渡します。

webservice ゾーンは、MLP を使用して restricted ゾーンのポート 80 で要求を待機し、webcontent のラベル付けされたゾーンからコンテンツを読み取ります。

webcontent ゾーンは動作可能状態にあって、その Web コンテンツを /export/home ファイルシステムに格納しています。このファイルシステムは、その他のすべてのラベル付けされたゾーンに自動的にマウントされます。ゾーンが動作可能状態である場合、そのゾーンで実行されているプロセスはありません。すなわち、基本的にゾーンは、webservice ゾーンに直接接続されたディスクドライブです。

Web ガードプロトタイプを構成するには、次のハイレベルタスクを実行します。

1. 安全な Web ブラウジング環境にラベルを構成するための label_encodings ファイルの変更

   デフォルトの label_encodings ファイルを更新して、2 つの新しいラベル (WEB GUARD SERVICE および WEB GUARD CONTENT) を構成します。103 ページの「label_encodings ファイルの変更」を参照してください。

2. トラステッドネットワークの構成

   restricted および webservice のラベル付けされたゾーンに、プライベート IP アドレスおよび MLP を構成します。106 ページの「トラステッドネットワークの構成」を参照してください。

3. Apache Web サーバーの構成

   public、restricted、および webservice ゾーンは、すべて Web サーバーを構成します。この例で使用される Web サーバーは Apache です。108 ページの「Apache Web サーバーの構成」を参照してください。

label_encodings ファイルの変更

デフォルトの label_encodings ファイルを更新して、2 つの新しいラベル (WEB GUARD SERVICE および WEB GUARD CONTENT) を構成します。デフォルトファイルの一部である SANDBOX ラベルは、WEB GUARD CONTENT ラベルとなるように変更します。WEB GUARD SERVICE ラベルを追加します。

label_encodings ファイルを /etc/security/tsol ディレクトリにインストールする必要があります。このファイルは、既存の Trusted Extensions インストールに追加してインストールできます。

更新したファイルを /etc/security/tsol ディレクトリにインストールしたら、次のようにして、新しい label_encodings ファイルをアクティブにします。

```
# svcadm restart svc:/system/labeld
```

この Web ガードプロトタイプで使用される label_encodings ファイルは、次のとおりです。

```
* ident    "@(#)label_encodings.simple    5.15    05/08/09 SMI"
*
* Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
* 使用はライセンス条項に従ってください。
*
* この例は、電子メールおよびプリンタ出力のラベル付けに関して、
* 実際のサイトの法的な情報保護要件を満たすラベルを指定する方法を
* 示します。ここに示すラベルは、ファイルおよびディレクトリに対する
* ユーザー認可上限ラベルおよび機密ラベルに基づく必須アクセス制御
```

* チェックを強制するためにも使用できます。

```
VERSION= Sun Microsystems, Inc. Example Version - 6.0. 2/15/05

CLASSIFICATIONS:

name= PUBLIC; sname= PUB; value= 2; initial compartments= 4;
name= CONFIDENTIAL; sname= CNF; value= 4; initial compartments= 4;
name= WEB GUARD; sname= WEB; value= 5; initial compartments= 0;
name= MAX LABEL; sname= MAX; value= 10; initial compartments= 0 4 5;

INFORMATION LABELS:

WORDS:

name= :; prefix;

name= INTERNAL USE ONLY; sname= INTERNAL; compartments= 1 ~2; minclass= CNF;
name= NEED TO KNOW; sname= NEED TO KNOW; compartments= 1-2 ~3; minclass= CNF;
name= RESTRICTED; compartments= 1-3; minclass= CNF;
name= CONTENT; compartments= 0 ~1 ~2 ~3; minclass= WEB;
name= SERVICE; compartments= 5; minclass= WEB;

REQUIRED COMBINATIONS:

COMBINATION CONSTRAINTS:

SENSITIVITY LABELS:

WORDS:

name= :; prefix;

name= INTERNAL USE ONLY; sname= INTERNAL; compartments= 1 ~2; minclass= CNF;
prefix= :

name= NEED TO KNOW; sname= NEED TO KNOW; compartments= 1-2 ~3; minclass= CNF;
prefix= :

name= RESTRICTED; compartments= 1-3; minclass= CNF; prefix= :

name= CONTENT; compartments= 0 ~1 ~2 ~3; minclass= WEB;

name= SERVICE; compartments= 5; minclass= WEB;

REQUIRED COMBINATIONS:

COMBINATION CONSTRAINTS:
```

```
CLEARANCES:

WORDS:

name= INTERNAL USE ONLY; sname= INTERNAL; compartments= 1 ~2; minclass= CNF;
name= NEED TO KNOW; sname= NEED TO KNOW; compartments= 1-2 ~3; minclass= CNF;
name= RESTRICTED; sname= RESTRICTED; compartments= 1-3; minclass= CNF;
name= CONTENT; compartments= 0 ~1 ~2 ~3; minclass= WEB;
name= SERVICE; compartments= 5; minclass= WEB;

REQUIRED COMBINATIONS:

COMBINATION CONSTRAINTS:

CHANNELS:

WORDS:

PRINTER BANNERS:

WORDS:

ACCREDITATION RANGE:

classification= PUB; all compartment combinations valid;
classification= WEB; all compartment combinations valid;
classification= CNF; all compartment combinations valid except: CNF

minimum clearance= PUB;
minimum sensitivity label= PUB;
minimum protect as classification= PUB;
```

* ローカルサイト定義とローカルで構成可能なオプション。

```
LOCAL DEFINITIONS:

default flags= 0x0;
forced flags= 0x0;

Default Label View is Internal;

Classification Name= Classification;
Compartments Name= Sensitivity;

Default User Sensitivity Label= PUB;
Default User Clearance= CNF NEED TO KNOW;
```

```
COLOR NAMES:

    label= Admin_Low;           color= #bdbdbd;

    label= PUB;                 color= blue violet;
    label= WEB SERVICE;         color= yellow;
    label= CNF;                 color= navy blue;
    label= CNF : INTERNAL USE ONLY;  color= blue;
    label= CNF : NEED TO KNOW;  color= #00bfff;
    label= CNF : RESTRICTED;    color= #87ceff;

    label= Admin_High;          color= #636363;
```

* ローカルサイト定義の終わり

label_encodings ファイルについての詳細は、『Solaris Trusted Extensions ラベルの管理』を参照してください。

トラステッドネットワークの構成

restricted および webservice ゾーンには、すでに共有している IP アドレスのほかに、プライベート IP アドレスを割り当てます。各プライベート IP アドレスはマルチレベルポートを構成し、制限されたラベルセットに関連付けられます。

次の表は、それぞれのラベル付けされたゾーンのネットワーク構成を示します。

ゾーン名	ゾーンラベル	ローカル IP アドレス	ホスト名	マルチレベルポート	セキュリティーラベルセット
restricted	CONFIDENTIAL :RESTRICTED	10.4.5.6	proxy	8080/tcp	PUBLIC
webservice	WEB GUARD SERVICE	10.1.2.3	webservice	80/tcp	CONFIDENTIAL :RESTRICTED
webcontent	WEB GUARD CONTENT	なし			

最初に、新しいゾーンを作成します。public ゾーンなどの既存のゾーンを複製できます。これらのゾーンを作成したら、zonecfg コマンドを使用して、表に示したアドレスのネットワークおよびローカルインタフェース名を追加します。

たとえば、次のコマンドは、10.4.5.6 の IP アドレスおよび bge0 インタフェースを restricted ゾーンに関連付けます。

```
# zonecfg -z restricted
add net
set address=10.4.5.6
set physical=bge0
end
exit
```

それぞれのラベル付けされたゾーンにIPアドレスおよびネットワークインタフェースを指定したら、Solaris管理コンソールを使用して表の残りの値を設定します。このツールを使用する場合、必ずScope=FilesおよびPolicy=TSOLのツールボックスを選択してください。

次の手順に従ってゾーン構成を終了します。

1. スーパーユーザーとしてSolaris管理コンソールを起動します。

   ```
   # &
   ```

2. 「ナビゲーション」パネルから「このコンピュータ」を選択し、「システム構成」をクリックします。
3. 「コンピュータおよびネットワーク」アイコンをクリックします。
4. 「コンピュータ」アイコンをクリックし、「アクションからコンピュータを追加」メニューを選択します。
5. proxyホストおよびwebserviceホストのホスト名およびIPアドレスを追加します。
6. 「ナビゲーション」パネルから「信頼できるネットワークゾーン」を選択します。

   列を拡張する必要のある場合があります。ゾーン名がリストに表示されない場合、「アクション」メニューから「ゾーン構成を追加」を選択します。
7. 各ゾーンにラベルを割り当て、「ローカルIPアドレスのMLP構成」フィールドに適切なポートおよびプロトコルを指定します。
8. 「ナビゲーション」パネルから「セキュリティーファミリ」アイコンをクリックし、「アクション」メニューから「テンプレートを追加」を選択します。

   表の情報に基づいてproxyホスト名およびwebservicesホスト名のテンプレートを追加します。

   a. テンプレート名の該当するホスト名を指定します。
   b. 「ホストタイプ」フィールドでCIPSOを指定します。
   c. 「最下位ラベル」および「最上位ラベル」フィールドで該当するゾーンラベルを指定します。
   d. 「セキュリティーラベルセット」フィールドで該当するセキュリティーラベルを指定します。
   e. 「明示的に割り当てられたホスト」タブをクリックします。

f. 「エントリを追加」セクションで、該当するローカル IP アドレスを各テンプレートに追加します。

9. Solaris 管理コンソールを終了します。

Solaris 管理コンソールを終了したら、影響があるゾーンを起動または再起動します。大域ゾーンで、新しいアドレスのルートを追加します。*shared-IP-addr* は共有 IP アドレスです。

```
# route add proxy shared-IP-addr
# route add webservice shared-IP-addr
```

Apache Web サーバーの構成

Apache Web サーバーのインスタンスは、public ゾーン、restricted ゾーン、および webservice ゾーンで実行されます。各ゾーンで、次のように /etc/apache/httpd.conf ファイルを変更します。

- public ゾーン – 次のように ServerName キーワードにサーバーの IP アドレスまたはホスト名を指定し、プロキシ構成を更新します。

  ```
  ServerName myserver

  ProxyRequests Off
  ProxyPass /demo http://proxy:8080/demo
  ProxyPassReverse /demo http://proxy:8080/demo
  ```

- restricted ゾーン – 待機プロキシポートおよびポートを指定します。さらに、次のように ServerName キーワードにこのゾーンの IP アドレスまたはホスト名を指定し、プロキシ構成を更新します。

  ```
  Listen proxy:8080
  Port 8080

  ServerName proxy

  ProxyRequests Off
  ProxyPass /demo http://webservice
  ProxyPassReverse /demo http://webservice
  ```

  Web コンテンツに対する要求の種類を制限するために、禁止用語フィルタなどの Web 要求のフィルタリングを設定することもできます。

- webservice ゾーン – 次のように ServerName キーワードにこのゾーンの IP アドレスまたはホスト名を指定し、DocumentRoot キーワードおよび <Directory> 要素 に Web コンテンツディレクトリの位置を指定します。

```
ServerName webservice

DocumentRoot "/zone/webcontent/export/home/www/htdocs"
<Directory "/zone/webcontent/export/home/www/htdocs">
```

ラベル付けされたゾーンごとに Apache Web サーバーの構成を更新したら、webcontent ゾーンの /export/home/www/htdocs ディレクトリに Web コンテンツを格納します。

/export/home/www/htdocs ディレクトリに demo ディレクトリを作成し、そのディレクトリにテストで使用する index.html ファイルを作成します。

起動時に、webservice ゾーンに lofs を使用することによって、/export/home ディレクトリは自動的にマウントされます。webcontent ゾーンは動作可能状態にする必要があるだけです。

```
# zoneadm -z webcontent ready
```

ゾーンが動作可能状態である場合、そのゾーンで実行されているプロセスはありません。ゾーンのファイルシステムは、webservice ゾーンによって読み取り専用でマウントできます。このように Web コンテンツにアクセスすると、コンテンツが変更されることがありません。

信頼できる Web ガードデモの実行

public ゾーンのブラウザから、または PUBLIC ラベルで実行されている遠隔ブラウザから、次の URL を入力します。

```
http://server-name/demo
```

webcontent ゾーンのデフォルトの index.html ファイルがブラウザに表示されます。

Web ガードフローはバイパスできません。webservice ゾーンの Web サーバーは、public ゾーンおよび遠隔ホストからパケットを受信できません。webcontent ゾーンが動作可能状態であるので、この Web コンテンツは変更できません。

下位レベルの信頼できないサーバーへのアクセス

場合によって、クライアントは、ラベル付けされていないシステムのサーバーにアクセスできなければなりません。「ラベル付けされていないシステム」は、Trusted Extensions ソフトウェアを実行しないシステムです。この場合、マルチレベルポートは、大域ゾーンまたはラベル付けされたゾーンで実行される特権サーバーに制限されるので使用できません。

たとえば、ブラウザが INTERNAL ゾーンで実行されているとします。tnrhdb データベースによって PUBLIC 機密ラベルが割り当てられた単一レベルネットワークで実行されている Web サーバーにアクセスしてみます。デフォルトでは、このようなアクセスは許可されません。それに対し、HTTP 要求を PUBLIC の Web サーバーに転送する特権プロキシーサーバーを記述できます。プロキシは、SO_MAC_EXEMPT と呼ばれる特別な Trusted Extensions ソケットオプションを使用します。このソケットオプションによって、信頼できない下位レベルのサービスに要求が送信され、サービスからの応答が要求元に返されます。

注 – SO_MAC_EXEMPT オプションを使用する場合、保護されないダウングレードチャネルになるので、十分に注意が必要です。呼び出し元プロセスの実効セットに PRIV_NET_MAC_EXEMPT 特権がない場合は、SO_MAC_EXEMPT オプションを設定できません。このようなプロセスでは、高位レベルのデータが低位レベルのサービスに漏れないように、その自身のデータフィルタリングポリシーを強制する必要があります。たとえば、単語が値として使用されないように、URL の不適切な箇所をプロキシで削除します。

次のコード抜粋は、connect.c にある wget コマンドの connect_to_ip() ルーチンの変更バージョンにおける SO_MAC_EXEMPT の使用方法を示します。SO_MAC_EXEMPT オプションの設定方法を明らかにするために、setsockopt() の呼び出しが追加されています。

```
int
connect_to_ip (const ip_address *ip, int port, const char *print)
{
  struct sockaddr_storage ss;
  struct sockaddr *sa = (struct sockaddr *)&ss;
  int sock;
  int on = 1;

  /* PRINT が NULL 以外の場合、PRINT は接続先のホスト名で
 "Connecting to..." 行を出力する。 */
  if (print)
    {
      const char *txt_addr = pretty_print_address (ip);
      if (print && 0 != strcmp (print, txt_addr))
    logprintf (LOG_VERBOSE, _("Connecting to %s|%s|:%d... "),
           escnonprint (print), txt_addr, port);
      else
    logprintf (LOG_VERBOSE, _("Connecting to %s:%d... "), txt_addr, port);
    }

  /* sockaddr 情報を SA に格納する。 */
  sockaddr_set_data (sa, ip, port);
```

```
  /* アドレスに対応するファミリのソケットを作成する。  */
  sock = socket (sa->sa_family, SOCK_STREAM, 0);
  if (sock < 0)
    goto err;

  if (setsockopt (sock, SOL_SOCKET, SO_MAC_EXEMPT, &on, sizeof (on)) == -1) {
    perror("setsockopt SO_MAC_EXEMPT");
  }

#if defined(ENABLE_IPV6) && defined(IPV6_V6ONLY)
  if (opt.ipv6_only) {
    /* エラーの場合、そのまま実行する... */
    int err = setsockopt (sock, IPPROTO_IPV6, IPV6_V6ONLY, &on, sizeof (on));
  }
#endif
```

◆ ◆ ◆ 第 9 章

9 Solaris Trusted Extensions ラベル API 用の実験的 Java バインディング

この章では、Solaris Trusted Extensions ソフトウェアに備えられているラベル API (アプリケーションプログラミングインタフェース) をミラー化する Java™ クラスおよびメソッドの実験的な設定について説明します。Trusted Extensions ラベル API の Java 実装は、ラベル対応アプリケーションの作成に使用することを目的とします。したがって、Trusted Extensions で提供されるラベル API のすべてが、Java 実装の一部に含まれるわけではありません。

これら実験的 Java API (Java バインディング) の提示では、Solaris Trusted Extensions の機能を Java の開発環境にどのように展開できるかを示します。

注意 – このような実験的 Java バインディングは、 Solaris Trusted Extensions ソフトウェアのサポート対象部分ではありません。

この章の内容は次のとおりです。

Java バインディングの概要

Java 言語は、セキュリティー保護されたマルチレベル領域で実行されるラベル対応アプリケーションの作成では、未開発のリソースです。これらの実験的 Java バインディングにより、システム監査ログの生成やシステムリソースの制御など、より多くのアプリケーションを開発するための基盤が提供されます。

Java 環境にプラットフォームサービスを追加すると、Java アプリケーションでマルチレベルの機密データを処理できるようになります。

Solaris Trusted Extensions では、ラベルデーモン `labeld` を介してラベルサービスが提供されます。このデーモンは、大域ゾーンおよびラベル付きゾーンで実行されるプロセスに使用できます。

この章で説明する Java バインディングは、一部の Solaris Trusted Extensions ラベル API の Java Native Interface (JNI™) 実装です。実験的 JNI コードで Solaris Trusted Extensions ラベルのライブラリ関数を呼び出して、ラベル機能の一部を Java 言語に拡張します。これらの Java クラスのコンストラクタおよびメソッドはC で記述された非公開 JNI インタフェースを呼び出し、呼び出された非公開 JNI インタフェースは Trusted Extensions API を呼び出します。たとえば、`SolarisLabel.dominates` メソッドで、C で記述された専用 JNI インターフェースを呼び出し、これを使用して `bldominates()` ルーチンを呼び出します。このような実験的 Java バインディングは、Java 2 Platform, Standard Edition 5.0 を使用して開発されました。JNI の詳細は、Java Native Interface のマニュアル (`http://java.sun.com/j2se/1.5.0/docs/guide/jni/`)を参照してください。

この実験的コードは、OpenSolaris™ の Web サイト (`http://www.opensolaris.org/os/community/security/projects/tx/`) の Solaris Trusted Extensions プロジェクトのページからダウンロードできます。

実験的 Java ラベルインタフェースの構造

Trusted Extensions ラベル API の JNI 実装により、次のように相互に関連するいくつかのラベル関連クラスが導入されます。

- `SolarisLabel` abstract クラス
  - `ClearanceLabel` サブクラス
  - `SensitivityLabel` サブクラス
- `Range` クラス

`SolarisLabel` abstract クラス

`SolarisLabel` abstract クラスは、Solaris Trusted Extensions ラベルに関連付けられた共通のネイティブメソッドの基盤を提供します。`SensitivityLabel` および `ClearanceLabel` サブクラスは、この abstract クラスからメンバーを継承します。機密ラベルおよび認可上限ラベルを作成するための static ファクトリも、この abstract クラスにより提供されます。

static ファクトリおよびメソッドは、エラーが検出されると例外をスローして、メッセージを表示せずに必須アクセス制御に関するエラーが発生していないことを確認します。

この abstract クラスで次の汎用メソッドを定義し、これを使用してラベルを比較したりラベルを文字列に変換したりします。

- `dominates`
- `equals`
- `setFileLabel`
- `strictlyDominates`
- `toColor`
- `toInternal`
- `toRootPath`
- `toString`
- `toText`
- `toTextLong`
- `toTextShort`

`equals`、`dominates` および `strictlyDominates` メソッドは、Solaris Trusted Extensions で現在使用可能な `blequal()`、`bldominates()`、および `blstrictdom()` ラベル API とよく似ています。`setFileLabel` メソッドは、Solaris Trusted Extensions で現在使用可能な `setflabel()` ルーチンとよく似ています。

`toText`、 `toInternal`、`toColor`などのその他のメソッドは、現在 Solaris Trusted Extensions で使用可能な `label_to_str()` ルーチンと機能が関連しています。これらのメソッドで、ラベルを特定の型の文字列に変換することができます。プロセスおよびオブジェクトのラベル関係によっては、ラベルを人間が読み取れる形式に変換するための実効セットで特権が必要となる場合があります。たとえば、Java 仮想マシン (JVM™) が優位でないラベルを変換するには、そのプロセスを `sys_trans_label` 特権を使って実行している必要があります。

`SolarisLabel` abstract クラスには次の static ファクトリも含まれます。

- `getClearanceLabel`
- `getFileLabel`
- `getSensitivityLabel`
- `getSocketPeer`

ラベルとして `getSensitivityLabel` または `getClearanceLabel`に渡す文字列は、次の形式のいずれかにすることができます。

- `PUBLIC` などの人間が読み取れる形式のラベル
- `0x0002-08-08` などの内部形式のラベル

保存やネットワーク接続上での伝送に適しているのは内部形式のラベルだけですが、これは、内部形式は実際のラベルを明らかにしないためです。詳細は、40 ページの「ラベルの読み取り可能バージョン」を参照してください。

`ClearanceLabel` および `SensitivityLabel` サブクラスは、`SolarisLabel` abstract クラスを拡張します。これらのサブクラスはそれぞれ、`SolarisLabel` abstract クラスで提供される共通メソッドを継承します。

ClearanceLabel サブクラス

ClearanceLabel サブクラスは SolarisLabel abstract クラスを拡張し、getMaximumおよび getMinimum メソッドを定義して、最小上限と最大下限をそれぞれに表す ClearanceLabel オブジェクトを返します。

SensitivityLabel サブクラス

SensitivityLabel サブクラスは SolarisLabel abstract クラスを拡張し、getMaximumおよび getMinimumメソッドを定義して、最小上限と最大下限をそれぞれに表す SensitivityLabel オブジェクトを返します。

SensitivityLabel サブクラスにより導入される次のメソッドにより、ラベル付けされたプリンタバナーページに適した情報が提供されます。

- toCaveats
- toChannels
- toFooter
- toHeader
- toProtectAs

Range クラス

Range クラスは、Java バージョンの Solaris Trusted Extensions ラベル範囲を表します。

このクラスで次の汎用メソッドを定義し、これを使用してラベル範囲内の上限および下限のラベルを取得して、ラベルが指定したラベル範囲内にあるかどうかを判別します。

- getLower
- getUpper
- inRange

Range クラスには次の static ファクトリも含まれており、範囲オブジェクトを作成します。

- getDeviceRange
- getLabelRange
- getUserRange

getDeviceRangeStaticファクトリと getUserRange static ファクトリは、それぞれ指定されたデバイスと指定されたユーザーの範囲に基づいて、範囲オブジェクトを作成します。getLabelRange static ファクトリは、範囲の上限と下限を指定したラベル範囲を作成することができます。

Java バインディング

Trusted Extensions ラベル API の Java 実装は、ラベル対応アプリケーションの作成に使用することを目的とします。したがって、Trusted Extensions で提供されるラベル API のすべてが Java 実装の一部に含まれるわけではありません。

この章で提示されている Java クラスおよびメソッドは、33 ページの「ラベル API」に示されている次の一般的なラベル機能によく似ています。

- Trusted Extensions システムの検出
- プロセス機密ラベルへのアクセス
- ラベルオブジェクトのためのメモリーの割り当てと解放
- ファイルのラベルの取得と設定
- ラベル範囲オブジェクトの取得
- ゾーンのラベルへのアクセス
- 遠隔ホストタイプの取得
- ラベルと文字列との変換
- ラベルオブジェクトの比較

Trusted Extensions システムの検出

これらの Java バインディングには、システムにラベルが付けられているかどうかを判別するメソッドは含まれません。むしろ、Trusted Extensions が有効でない場合にはライブラリでのロードが失敗します。

プロセス機密ラベルへのアクセス

これらの Java バインディングには、プロセスのラベルを取得するメソッドは含まれません。Trusted Extensions では、ラベル付けされたゾーンで実行されるプロセスに、そのゾーンと同じラベルがあります。

ラベルオブジェクトのためのメモリーの割り当てと解放

これらの Java バインディングでは、Java の「ガベージコレクション」機能を利用します。これによって、36 ページの「ファイルのラベルの取得と設定」で説明されているラベル API に対して行うように、ラベルオブジェクトによって使用されたメモリーを明示的に解放する必要がありません。

ファイルのラベルの取得と設定

これらの Java バインディングでは、Java File オブジェクトを使用して、ファイルラベルの取得および設定を行います。getFileLabel static ファクトリを使用して、ファイルの File オブジェクトからラベルを取得します。ファイルラベルを別の指定ラベルに設定するには、 setFileLabel メソッドをファイルの File オブジェクトに対して使用します。

ファイルの機密ラベルの取得に加えて、getSocketPeer static ファクトリは、ソケットのピア終端用に機密ラベルの取得を可能にします。

getFileLabel static ファクトリおよび setFileLabel メソッドは、それぞれ getlabel() システムコールと setflabel() ルーチンにそれぞれ対応します。詳細は、36 ページの「ファイルのラベルの取得と設定」および getlabel(2) と setflabel(3TSOL) のマニュアルページを参照してください。

次の記述には、static ファクトリおよびメソッドのプロトタイプ宣言が含まれます。

`public static SensitivityLabel getFileLabel(java.io.File file)`
: getFileLabel static ファクトリでは、*file* によって指定された Java File オブジェクトのラベルを取得します。

`public static SensitivityLabel getSocketPeer(java.net.Socket socket)`
: getSocketPeer static ファクトリは、指定されたソケット socket から機密ラベルオブジェクト、*socket* を取得します。

  次のコードフラグメントは、ソケット *s* の機密ラベルオブジェクトを取得します。

```
SensitivityLabel sl = SolarisLabel.getSocketPeer(s);
```

  次のコーディグ例は、サーバーソケットをポート 9090 上に作成し、使用可能な接続のピア終端の機密ラベルを取得する方法を示します。このコーディング例では、取得されたソケットピアラベルに関して、内部形式、人間が読み取れる形式、色、およびルートパスも出力されます。

```
import java.io.*;
import java.net.*;
import solarismac.*;

public class ServerSocketTest
{

    public static void main (String args[]) {

        System.out.println("ServerSocketTest Start");

        CreateListner();
```

```
            System.out.println("ServerSocketTest End");

        }

        /*
         * Listen for connections on port then print the peer connection label.
         * You can use telnet host 9090 to create a client connection.
         */
        private static void CreateListner() {
            int port = 9090;

            ServerSocket acceptSocket;
            Socket s;
            try {
                System.out.println("Creating ServerSocket on port " + port);

                acceptSocket = new ServerSocket(port);

                System.out.println("ServerSocket created, waiting for connection");

                s = acceptSocket.accept();

                /*
                 * Get the Sensitivity Label for the peer end of the socket.
                 */
                SensitivityLabel socksl = SolarisLabel.getSocketPeer(s);

                System.out.println("Client connected...");
                System.out.println(" toInternal: " + socksl.toInternal());
                System.out.println(" toText: " + socksl.toText());
                System.out.println(" toString: " + socksl.toString());
                System.out.println(" toColor: " + socksl.toColor());
                System.out.println(" toRootPath: " + socksl.toRootPath());
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

`public static void setFileLabel(java.io.File file, SensitivityLabel label)`
`setFileLabel` メソッドで、指定したファイルの機密レベルを指定したラベルに変更します。ファイルの機密ラベルが変わると、新しいラベルに対応するゾーンにファイルが移動します。ファイルは、ほかのゾーンのルートに相対的な新しいパス名に移動します。

たとえば、`setFileLabel` メソッドを使用して、ファイル `/zone/internal/documents/designdoc.odt` のラベルを `INTERNAL` から `RESTRICTED` に

変更すると、このファイルの新しいパスは /zone/restricted/documents/designdoc.odt になります。移動先のディレクトリが存在しない場合、ファイルは移動しません。

次のコードフラグメントは、ファイルのラベルをどのような方法で変更すべきかを示しています。

```
SolarisLabel.setFileLabel(new File("/zone/internal/documents/designdoc.odt"),
    SolarisLabel.getSensitivityLabel("RESTRICTED"));
```

ファイルの機密ラベルを変更すると、元のファイルは削除されます。移動元と移動先のファイルシステムが同じ基礎となるファイルシステムからループバックマウントされる場合は例外です。この場合、ファイルの名前が変更されます。

Java 仮想マシンは、ファイルのラベル変更に適した特権 (file_upgrade_sl または file_downgrade_sl) で実行されている必要があります。

特権の設定に関する詳細は、『Solaris セキュリティーサービス開発ガイド』の第 2 章「特権付きアプリケーションの開発」を参照してください。 setflabel(3TSOL) のマニュアルページも参照してください。

ラベル範囲オブジェクトの取得

getLabelRange static ファクトリで、ラベル範囲オブジェクトを作成します。getUserRange および getDeviceRange static ファクトリでは、ユーザーおよびデバイスのラベル範囲オブジェクトをそれぞれ取得します。getUpper および getLower メソッドを使用して、範囲の上位ラベルおよび下位ラベルをそれぞれ取得します。 さらに、inRangeメソッドを使用して、指定したラベルが範囲内にあるかどうかを判別します。inRange の詳細は、126 ページの「ラベルオブジェクトの比較」 を参照してください。

getUserRange および getDeviceRange static ファクトリは、 getuserrange() および getdevicerange() ルーチンに対応します。詳細は、37 ページの「ラベル範囲の取得」 および getdevicerange(3TSOL) のマニュアルページを参照してください。

次のコンストラクタおよびメソッドの記述には、各コンストラクタのプロトタイプ宣言が含まれています。

public static Range getDeviceRange(java.lang.String device)
: getDeviceRange static ファクトリでは、ユーザーが割り当て可能なデバイスのラベル範囲を取得します。デバイスにラベル範囲を指定しない場合のデフォルト範囲は、上限が ADMIN_HIGH、下限が ADMIN_LOW です。

  list_devices コマンドを使用して、デバイスのラベル範囲を表示できます。list_devices(1) および getdevicerange(3TSOL) のマニュアルページを参照してください。

`public static <L extends SolarisLabel,U extends SolarisLabel> Range getLabelRange(L lower, U upper)`
: `getLabelRange` static ファクトリでラベル範囲を作成します。static ファクトリでは、パラメータとして範囲の下限値および上限、すなわち認可上限 を利用します。*upper* が *lower* より優位でない場合は、例外がスローされます。

`public L getLower()`
: `getLower` メソッドは範囲内の下位の部分を返します。

`public U getUpper()`
: `getUpper`メソッドは範囲内の上位の部分を返します。

`public static Range getUserRange(java.lang.String user)`
: `getUserRange` static ファクトリは、指定したユーザーのラベル範囲を取得します。ユーザーがマルチレベルデスクトップにログインするときに、範囲の下限がラベルとして使用されます。範囲の上限、すなわち認可上限は、ラベル付けされたワークスペースにユーザーが割り当てることができる使用可能なラベルに対する上限として使用されます。

  ユーザーのラベル範囲のデフォルト値は `label_encodings` ファイルで指定します。その値は `user_attr` ファイルによって上書きできます。

  詳細は、`getuserrange(3TSOL)` のマニュアルページを参照してください。

ゾーンのラベルへのアクセス

次の記述には、メソッドのプロトタイプ宣言が含まれます。

`public final java.lang.String toRootPath()`
: このメソッドは、指定した機密ラベルのゾーンのルートパス名を返します。

  次のコーディング抜粋は、`PUBLIC` 機密ラベルのルートパスを取得する方法を示します。

```
SensitivityLabel sl = SolarisLabel.getSensitivityLabel("PUBLIC");
System.out.println("toRootPath: " + sl.toRootPath();
```

  無効なラベルが指定されるか、または指定したラベルにゾーンが設定されていない場合、このメソッドは `java.io.IOException` をスローします。

  このメソッドは `getzonerootbylabel()` ルーチンによく似ています。`getzonerootbylabel(3TSOL)` のマニュアルページを参照してください。38 ページの「ゾーンのラベルへのアクセス」も参照してください。

遠隔ホストタイプの取得

Trusted Extensions ラベル API の Java 実装には、遠隔ホストタイプ取得用のインタフェースは含まれません。

ラベルと文字列との変換

SolarisLabel abstract クラスには、ラベルと文字列間の変換に使用するメソッドが含まれており、そのメソッドのサブクラスによって継承されます。

これらのメソッドで、ラベルの内部表現 (m_label_t) を String オブジェクトに変換します。

toInternal メソッドを使用して、格付け名を表示しない文字列にラベルを変換できます。この形式は、共通オブジェクトでのラベルの格納に適します。

Java 仮想マシンの実行は、変換されるラベルよりも優位であることが必要です。つまり、sys_trans_label 特権を持つ必要があります。label_to_str(3TSOL) のマニュアルページを参照してください。

一部のラベル値は、label_encodings ファイル内のデータに基づきます。

次のメソッドは label_to_str() ルーチンによく似ています。label_to_str(3TSOL) のマニュアルページを参照してください。

public final java.lang.String toColor()
: このメソッドは SolarisLabel オブジェクトの色を返します。この値は HTML での使用に適しています。

public final java.lang.String toInternal()
: このメソッドは、公開オブジェクトへの格納に安全なラベルの内部表現を返します。内部変換は、あとでその同じ値に解析することができます。これは toString メソッドの使用方法と同じです。

  これらの 2 つのメソッドでは、エラーの処理方法が異なります。toInternal メソッドでエラーが検出された場合には、java.io.IOException が返されます。一方、toString メソッドでエラーが検出された場合は、NULL が返されます。

public java.lang.String toString()
: このメソッドは、ラベルの 内部 16 進数バージョンを文字列形式で返します。これは、toInternal メソッドの使用方法と同じです。

  これらの 2 つのメソッドでは、エラーの処理方法が異なります。toString メソッドでエラーが検出された場合は、NULL が返されます。ただし、 toInternal メソッドでエラーが検出された場合には、java.io.IOException が返されます。

`public java.lang.String toText()`
このメソッドは、SolarisLabel オブジェクトの人間が読み取れる文字列を長形式で返します。

`public java.lang.String toTextLong()`
このメソッドは、SolarisLabel オブジェクトの人間が読み取れる文字列を長形式で返します。

`public java.lang.String toTextShort()`
このメソッドは、SolarisLabel オブジェクトの人間が読み取れる文字列を長形式で返します。

次のメソッドで、マルチレベルのプリンタバナーページの作成に適したラベル変換を実行します。これらのメソッドは、label_to_str() ルーチンの一部の機能とよく似ています。label_to_str(3TSOL) および m_label(3TSOL) のマニュアルページを参照してください。

`public java.lang.String toCaveats()`
このメソッドは、バナーページの警告セクションの作成に適した、人間が読み取れるテキスト文字列を返します。

このメソッドは SensitivityLabel オブジェクトにのみ使用可能であり、ClearanceLabel オブジェクトには使用できません。

`public java.lang.String toChannels()`
このメソッドは、バナーページの処理チャネルセクションに適した、人間が読み取れるテキスト文字列を返します。

このメソッドは SensitivityLabel オブジェクトにのみ使用可能であり、ClearanceLabel オブジェクトには使用できません。

`public java.lang.String toFooter()`
このメソッドは、機密ラベルとしての使用に適した、人間が読み取れるテキスト文字列を返します。この機密ラベルは、末尾のページの下部に表示されます。

このメソッドは SensitivityLabel オブジェクトにのみ使用可能であり、ClearanceLabel オブジェクトには使用できません。

`public java.lang.String toHeader()`
このメソッドは、機密ラベルとしての使用に適した、人間が読み取れるテキスト文字列を返します。この機密ラベルは、末尾のページの上部に表示されます。

このメソッドは SensitivityLabel オブジェクトにのみ使用可能であり、ClearanceLabel オブジェクトには使用できません。

`public java.lang.String toProtectAs()`
このメソッドは、機密ラベルとしての使用に適した、人間が読み取れるテキスト文字列を返します。この機密ラベルは、のページの下部に表示されます。

このメソッドは SensitivityLabel オブジェクトにのみ使用可能であり、ClearanceLabel オブジェクトには使用できません。

例 9–1　Java バインディングを使用したバナーページの作成

次のサンプルコーディグは、Java バインディングを使用して52 ページの「プリンタバナー情報の取得」で説明されているものと同様のバナーページを作成する方法を示します。

```
import solarismac.*;
import java.io.*;

/*
 * Banner page example
 */
public class PrintTest1
{

   public static void main (String args[]) {

      try {

         // Pick a valid label using the label_encodings.example
         SensitivityLabel sl = SolarisLabel.getSensitivityLabel("TOP SECRET A B SA");

         // "Protect as classification"
         System.out.println(sl.toHeader());
         System.out.println();

         // "Protect as classification plus compartments"
         System.out.println("This output must be protected as:");
         System.out.println(sl.toProtectAs());
         System.out.println("unless manually reviewed and downgraded.");
         System.out.println();

         // Handling instructions specified in PRINTER BANNERS
         System.out.println(sl.toCaveats());
         System.out.println();

         // Handling instructions specified in CHANNELS
         System.out.println(sl.toChannels());
         System.out.println();

         // "Protect as classification"
         System.out.println(sl.toFooter());
         System.out.println();

      } catch (Exception e) {
```

例 9–1 Java バインディングを使用したバナーページの作成 (続き)

```
            e.printStackTrace();
        }
    }
}
```

TOP SECRET A B SA のプロセスラベルの場合、テキスト出力は次のようになります。

```
        TOP SECRET

This output must be protected as:

TOP SECRET A B SA

unless manually reviewed and downgraded.

   (FULL SA NAME)
   HANDLE VIA (CH B)/(CH A) CHANNELS JOINTLY

        TOP SECRET
```

toText、toInternal および toColor などのメソッドでは、文字列からラベルに変換しません。文字列を機密ラベルまたは認可上限ラベルに変換するには、getSensitivityLabel または getClearanceLabel static ファクトリを個別に呼び出す必要があります。次の static ファクトリは str_to_label() ルーチンによく似ています。str_to_label(3TSOL) および m_label(3TSOL) のマニュアルページを参照してください。

public static ClearanceLabel getClearanceLabel(java.lang.String label)
この static ファクトリでは、指定された文字列から認可上限ラベルを作成します。次の例では、ラベル名およびラベルの内部 16 進数名に基づいて新しい認可上限ラベルを作成します。

```
ClearanceLabel cl = SolarisLabel.getClearanceLabel("PUBLIC");
ClearanceLabel cl = SolarisLabel.getClearanceLabel("0x0002-08-08");
```

public static SensitivityLabel getSensitivityLabel(java.lang.String label)
この static ファクトリでは、指定した文字列から機密ラベルを作成します。次の例では、ラベル名およびラベルの内部 16 進数名に基づいて新しい機密ラベルを作成します。

```
SensitivityLabel sl = SolarisLabel.getSensitivityLabel("PUBLIC");
SensitivityLabel sl = SolarisLabel.getSensitivityLabel("0x0002-08-08");
```

ラベルオブジェクトの比較

次の equals、dominates、および strictlyDominates メソッドを使用してラベルを比較します。これらのメソッドは blequal()、bldominate()、および blstrictdom() ルーチンに対応します。inRange メソッドを使用して、指定したラベル範囲内にラベルがあるかどうかを判別します。このメソッドは blinrange() ルーチンに対応します。これらのメソッドで *label* が指すのは、格付けおよび機密ラベルまたは認可上限ラベルのコンパートメントのセットです。詳細は、41 ページの「ラベルの比較」および blcompare(3TSOL) のマニュアルページを参照してください。

`public boolean dominates(SolarisLabel label)`
dominates メソッドで 2 つのラベルを比較して、一方のラベルがもう一方のラベルより優位であるかどうかを判別します。

次のサンプルコーディングは、どのように比較ができるかを示します。CNF : INTERNAL ラベルが比較されて、PUBLIC ラベルに対する優位性がチェックされます。

```
SensitivityLabel sl = SolarisLabel.getSensitivityLabel("CNF : INTERNAL");
boolean isDominant = sl.dominates(SolarisLabel.getSensitivityLabel("PUBLIC"));
```

`public boolean equals(java.lang.Object obj)`
equals ソッドで 2 つのラベルを比較して、それらが等位であるかどうかを判別します。

次のサンプルコーディングは、どのように比較ができるかを示します。

```
SensitivityLabel sl = SolarisLabel.getSensitivityLabel("CNF : INTERNAL");
boolean isSame = sl.equals(SolarisLabel.getSensitivityLabel("PUBLIC"));
```

`public boolean Range inRange(SensitivityLabel label)`
The inRange メソッドで、指定したラベルが範囲内にあるかどうかを判別します。このメソッドは Range クラスに属します。

次のコードフラグメントは、ファイルがユーザーの認可上限範囲内であるかどうかを検証できる方法を示します。

```
import solarismac.*;
import java.io.*;

public class Example1
{
    public static void main (String args[]) {

        try {
        Range range;

            range = Range.getUserRange("jweeks");
```

```
                SensitivityLabel fsl =
                    SolarisLabel.getFileLabel(new File("/etc/passwd"));
                boolean isInRange;

                isInRange = Range.inRange(fsl);

                if (isInRange)
                    System.out.println("File is within user's range");
                else
                    System.out.println("File is not within user's range");

            }
            catch (Exception e) {
                e.PrintStackTrace();
            }
        }
    }
```

`public boolean strictlyDominates(SolarisLabel label)`
`strictlyDominates` メソッドで 2 つのラベルを比較して、一方のラベルがもう一方のラベルよりも厳密に優位であるかどうかを判別します。一方のラベルがもう一方のラベルよりも厳密に優位である場合は、もう一方のラベルよりも優先されますが、等位ではありません。

次のサンプルコーディングは、どのように比較ができるかを示します。`CNF : INTERNAL` ラベルが比較されて、`PUBLIC` ラベルに対する完全な優位性がチェックされます。

```
SensitivityLabel sl = SolarisLabel.getSensitivityLabel("CNF : INTERNAL");
boolean isStrictlyDominant =
    sl.strictlyDominates(SolarisLabel.getSensitivityLabel("PUBLIC"));
```

ラベル関係の詳細は、17 ページの「ラベル関係」を参照してください。

`getMaximum` および `getMinimum` メソッドを使用して、指定した範囲の最小上限および最大下限をそれぞれ判別します。これらのメソッドで、`blmaximum()` および `blminimum()` ルーチンの機能をミラー化します。詳細は、41 ページの「ラベルの比較」 および `blminmax(3TSOL)` のマニュアルページを参照してください。

たとえば、この `getMaximum` を使用して、ラベル付けされた 2 つのオブジェクトの情報を組み合わせ、新たに別のオブジェクトを作成してラベル付けするときに使用するラベルを決定します。新しいオブジェクトのラベルは、ラベル付けされた元の 2 つのオブジェクトよりも優位となります。各メソッドは、`ClearanceLabel` および `SensitivityLabel` サブクラスで定義されます。

`public ClearanceLabel getMaximum(ClearanceLabel bounding)`
`getMaximum` メソッドは、ユーザーが指定する 2 つのラベルオブジェクトよりも優位である、最下ラベルとなる新しい認可上限ラベルオブジェクトを作成します。

結果として得られるオブジェクトは、その範囲の最小上限です。getMaximum は、認可上限ラベルの内部形式でオブジェクトを返します。

public ClearanceLabel getMinimum(ClearanceLabel bounding)
getMinimum メソッドは、ユーザーが指定する 2 つのラベルのどちらに対しても優位でない、最大ラベルとなる新しい認可上限ラベルオブジェクトを作成します。結果として得られるオブジェクトは、その範囲の最大下限です。getMinimum は、認可上限ラベルの内部形式でオブジェクトを返します。

public SensitivityLabel getMaximum(SensitivityLabel bounding)
getMaximum メソッドは、ユーザーが指定する 2 つのラベルオブジェクトよりも優位である、最下ラベルとなる新しい機密ラベルオブジェクトを作成します。結果として得られるオブジェクトは、その範囲の最小上限です。getMaximum は、機密ラベルの内部形式でオブジェクトを返します。

public SensitivityLabel getMinimum(SensitivityLabel bounding)
getMinimum メソッドは、ユーザーが指定する 2 つのラベルのどちらに対しても優位でない、最上位ラベルとなる新しい機密ラベルオブジェクトを作成します。結果として得られるオブジェクトは、その範囲の最大下限です。getMinimum は、機密ラベルの内部形式でオブジェクトを返します。

次の表は、 getMaximum および getMinimum メソッドからのラベルの入力および出力を示します。

表 9–1 getMinimum および getMaximum メソッド

入力ラベル	getMinimum 出力	getMaximum 出力
SECRET A B TOP SECRET A B SA SB CC	SECRET A B	TOP SECRET A B SA SB CC
SECRET A B TOP SECRET A SA CC	SECRET A	TOP SECRET A B SA CC
SECRET A B TOP SECRET	SECRET	TOP SECRET A B
SECRET A TOP SECRET B	SECRET	TOP SECRET A B

◆ ◆ ◆ 付 録 A

プログラマーズリファレンス

この付録では、Solaris Trusted Extensions ソフトウェアを使用する環境において、ラベル対応アプリケーションの開発、テスト、およびリリースに関する情報がある場所を示します。

この付録の内容は次のとおりです。

Trusted Extensions のマニュアルページ

`Intro(3TSOL)` のマニュアルページに、Trusted Extensions が設定されたシステムの概要が示されています。Trusted Extensions のすべてのマニュアルページは、Sun のマニュアル Web サイト (`http://www.sun.com/documentation/`)および『Solaris Trusted Extensions リファレンスマニュアル』で参照できます。

ヘッダーファイルの場所

ほとんどの Trusted Extensions ヘッダーファイルは、`/usr/include/tsol` ディレクトリおよび `/usr/include/sys/tsol` ディレクトリにあります。その他のヘッダーファイルの場所を次の表に示します。

ヘッダーファイルと場所	インタフェースのカテゴリ
/usr/dt/include/Dt/label_clipping.h	フォントリストによる X11 ラベル変換とラベルクリッピング
/usr/dt/include/Dt/ModLabel.h	ラベルビルダー
/usr/openwin/include/X11/extensions/Xtsol.h	X ウィンドウシステム
/usr/include/libtsnet.h	トラステッドネットワークライブラリ
/usr/include/bsm/libbsm.h	監査ライブラリ

インタフェース名およびデータ構造体名に使用される略号

Trusted Extensions インタフェース名およびデータ構造体名の多くには、次の略号が使用されます。これらの名前の略号は、インタフェースまたは構造体の目的の判別に役立ちます。

表 A–1 Trusted Extensions API によって使用される名前の略号

略号	名前
attr	属性
b	2 進数
clear	認可上限
ent	エントリ
f	ファイル
fs	ファイルシステム
h	16 進数
l	レベル、ラベル、またはシンボリックリンク
lbuild	ラベルビルダー
prop	プロパティー
r	再入可能
res	リソース
s	文字列
sec	セキュリティー

表 A–1　Trusted Extensions API によって使用される名前の略号　*(続き)*

略号	名前
sl	機密ラベル
tp	トラステッドパス
tsol	Trusted Extensions
xtsol	Trusted X11 サーバー

アプリケーションの開発、テスト、およびデバッグ

メインシステムのセキュリティーポリシーがソフトウェアバグおよびコードの不備によって損なわれないように、独立した開発システムでアプリケーションを開発、テスト、デバッグする必要があります。

次のガイドラインに従ってください。

- 余分なデバッグコード、特に文書化されていない機能を提供するコード、およびセキュリティーチェックをバイパスするコードを削除します。
- アプリケーションデータの操作に関して、セキュリティー上の問題がないかをインストール前に管理者が検査できるように、操作を簡単にします。
- すべてのプログラミングインタフェースについて、リターンコードをテストします。呼び出しが成功しない場合、予期しない結果になることがあります。予期しないエラー条件が発生した場合、アプリケーションは常に終了しなければなりません。
- すべての機密ラベルで、およびアプリケーションを実行すると予想されるすべての役割から、アプリケーションを実行することによってすべての機能をテストします。
  - プログラムが役割によってではなく、一般ユーザーによって実行される場合、プログラムが実行されるラベルでコマンド行からプログラムを起動します。
  - プログラムが役割によって実行される場合、大域ゾーンでコマンド行からプログラムを起動するか、プログラムが実行されるラベルでユーザー役割から起動します。
- アプリケーションが必要とするすべての特権がアプリケーションにあるかを確認するために、特権デバッグモードですべての機能をテストします。この種のテストでは、アプリケーションが実行すべきでない特権タスクをアプリケーションが実行しようとしているかどうかも確認します。
- 特権を使用することによって起きる、セキュリティーに対する影響を確認します。アプリケーションが特権を使用することによってシステムセキュリティーが損なわれないようにします。
- 適切な特権ブラケットを確認してそれに従います。

『Solaris セキュリティーサービス開発ガイド』を参照してください。

- 特権アプリケーションをテストするために SUNWspro debugger または dbx コマンドを使用する場合、それを実行プロセスにアタッチする前に、デバッガを起動します。コマンド名を引数としてデバッガを起動することはできません。

アプリケーションのリリース

十分にテストおよびデバッグしたアプリケーションを、アプリケーション統合のためにシステム管理者に提出します。アプリケーションの提出は、CDE アクションまたはソフトウェアパッケージとして行えます。アプリケーションが特権を使用する場合、システム管理者はアプリケーションのソースコードおよび提出されたセキュリティー情報を評価する必要があります。この評価によって、特権の使用がシステムセキュリティーを損なわないことを確認します。

注意 – アプリケーションが使用する新しい監査イベント、監査クラス、X ウィンドウシステムプロパティーについてシステム管理者に通知してください。システム管理者は、これらを適切なファイルに追加する必要があります。詳細は、第 6 章を参照してください。

CDE アクションの作成

CDE アクションは、ユーザーまたは役割によってワークスペースから起動されます。このアクションは、ユーザーまたは役割のプロファイルに割り当てられている特権を継承します。「*CDE* アクション」は、アプリケーションを実行してデータファイルを開くなどのデスクトップタスクを自動化するアプリケーションマクロまたは API のように機能する命令のセットです。Trusted Extensions が設定されたシステムでは、アプリケーションは CDE アクションとしてワークスペースから起動されます。CDE アクションの作成方法については、『Solaris 共通デスクトップ環境 上級ユーザ及びシステム管理者ガイド』を参照してください。

注 – CDE アクションを作成する場合、`f.exec` ではなく `f.action` を作成します。`f.exec` は、すべての特権を持つスーパーユーザーとしてプログラムを実行します。

システム管理者は、CDE アクションを適切なプロファイルに入れることによって、必要な特権を CDE アクションに割り当てます。システム管理者に対して、プログラムが使用する特権をリストし、アプリケーションが実行されるラベルを示し、必要な実効ユーザー ID またはグループ ID を指定する必要があります。システム管理者は、特権および実効ユーザー ID またはグループ ID をプロファイル内の CDE アクションに割り当てます。

ソフトウェアパッケージの作成

ソフトウェアパッケージを作成するには、『Application Packaging Developer's Guide』を参照してください。パッケージインストールの問題をデバッグするには、『Solaris のシステム管理 (上級編)』の第 14 章「ソフトウェアの問題解決 (概要)」を参照してください。

◆ ◆ ◆ 付 録 B

Solaris Trusted Extensions API リファレンス

この付録では、アプリケーションプログラミングインタフェース (API) をリストし、その使用法の相互参照を示します。宣言はセキュリティー項目ごとにまとめられています。

この付録の内容は次のとおりです。

プロセスセキュリティー属性フラグ API

次の Solaris API は Trusted Extensions パラメータを受け入れます。

- `uint_t getpflags(uint_t flag);`
- `int setpflags(uint_t flag, uint_t value);`

ラベル API

ラベル API は、第 2 章で概説されています。コーティング例は、第 3 章にありあす。詳しく説明されている例は、第 4 章にあります。

次に、ラベル関連 API のタイプをリストし、各タイプのルーチンおよびシステムコールのプロトタイプ宣言を示します。

- `label_encodings` ファイルへのアクセス
  - `m_label_t *m_label_alloc(const m_label_type_t label_type);`
  - `int m_label_dup(m_label_t **dst, const m_label_t *src);`
  - `void m_label_free(m_label_t *label);`
  - `int label_to_str(const m_label_t *label, char **string, const m_label_str_t conversion_type, uint_t flags);`
- レベル関係の比較
  - `int blequal(const m_label_t *level1, const m_label_t *level2);`
  - `int bldominates(const m_label_t *level1, const m_label_t *level2);`
  - `int blstrictdom(const m_label_t *level1, const m_label_t *level2);`
  - `int blinrange(const m_label_t *level, const brange_t *range);`
  - `void blmaximum(m_label_t *maximum_label, const m_label_t *bounding_label);`
  - `void blminimum(m_label_t *minimum_label, const m_label_t *bounding_label);`
- ラベル範囲へのアクセス
  - `m_range_t *getuserrange(const char *username);`
  - `blrange_t *getdevicerange(const char *device);`
- ゾーンのラベルへのアクセス
  - `char *getpathbylabel(const char *path, char *resolved_path, size_t bufsize, const m_label_t *sl);`
  - `m_label_t *getzonelabelbyid(zoneid_t zoneid);`
  - `m_label_t *getzonelabelbyname(const char *zonename);`
  - `zoneid_t *getzoneidbylabel(const m_label_t *label);`
  - `char *getzonerootbyid(zoneid_t zoneid);`
  - `char *getzonerootbylabel(const m_label_t *label);`
  - `char *getzonerootbyname(const char *zonename);`
- 遠隔ホストタイプの取得
  - `tsol_host_type_t tsol_getrhtype(char *hostname);`
- 機密ラベルへのアクセスと修正

- `int fgetlabel(int fd, m_label_t *label_p);`
- `int getlabel(const char *path, m_label_t *label_p);`
- `int setflabel(const char *path, const m_label_t *label_p);`
- `int getplabel(m_label_t *label_p);`
- `int label_to_str(const m_label_t *label, char **string, const m_label_str_t conversion_type, uint_t flags);`
- `int str_to_label(const char *string, m_label_t **label, const m_label_type_t label_type, uint_t flags, int *error);`

ラベルクリッピング API

ラベルクリッピング API については、第 6 章を参照してください。

```
int label_to_str(const m_label_t *label, char **string,
    const m_label_str_t conversion_type, uint_t flags);
```

RPC API

Trusted Extensions には、遠隔手続き呼び出し (RPC) 用のインタフェースがありません。Trusted Extensions と機能するように RPC インタフェースが変更されています。理論的な説明は、第 5 章を参照してください。`getpeerucred()` および `ucred_getlabel()` ルーチンを使用する例は、第 4 章を参照してください。

ラベルビルダー API

ラベルビルダーのユーザーインタフェースについては、第 7 章を参照してください。

- `ModLabelData *tsol_lbuild_create(Widget widget, void (*event_handler)() ok_callback, lbuild_attributes extended_operation, ..., NULL);`
- `void tsol_lbuild_destroy(ModLabelData *lbdata);`
- `void *tsol_lbuild_get(ModLabelData *lbdata, lbuild_attributes extended_operation);`
- `void tsol_lbuild_set(ModLabelData *lbdata, lbuild_attributes extended_operation, ..., NULL);`

トラステッド X ウィンドウシステム API

トラステッド X ウィンドウシステム API については、第 6 章を参照してください。

- Status XTSOLgetResAttributes(Display *display, XID object, ResourceType type, XTSOLResAttributes *winattrp);
- Status XTSOLgetPropAttributes(Display *display, Window window, Atom property, XTSOLPropAttributes *propattrp);
- Status XTSOLgetClientAttributes(Display *display, XID windowid, XTsolClientAttributes *clientattrp);
- Status XTSOLgetResLabel(Display *display, XID object, ResourceType type, m_label_t *sl);
- Status XTSOLsetResLabel(Display *display, XID object, ResourceType type, m_label_t *sl);
- Status XTSOLgetResUID(Display *display, XID object, ResourceType type, uid_t *uidp);
- Status XTSOLsetResUID(Display *display, XID object, ResourceType type, uid_t *uidp);
- Status XTSOLgetPropLabel(Display *display, Window window, Atom property, m_label_t *sl);
- Status XTSOLsetPropLabel(Display *display, Window window, Atom property, m_label_t *sl);
- Status XTSOLgetPropUID(Display *display, Window window, Atom property, uid_t *uidp);
- Status XTSOLsetPropUID(Display *display, Window window, Atom property, uid_t *uidp);
- Status XTSOLgetWorkstationOwner(Display *display, uid_t *uidp);
- Status XTSOLsetWorkstationOwner(Display *display, uid_t *uidp);
- Status XTSOLsetSessionHI(Display *display, m_label_t *sl);
- Status XTSOLsetSessionLO(Display *display, m_label_t *sl);
- Status XTSOLMakeTPWindow(Display *display, Window *w);
- Bool XTSOLIsWindowTrusted(Display *display, Window *window);
- Status XTSOLgetSSHeight(Display *display, int screen_num, int *newheight);
- Status XTSOLsetSSHeight(Display *display, int screen_num, int newheight);
- Status XTSOLsetPolyInstInfo(Display *display, m_label_t sl, uid_t *uidp, int enabled);

Trusted Extensions パラメータを使用する Solaris ライブラリルーチンとシステムコール

次の Solaris インタフェースは、Trusted Extensions パラメータを含むか、Trusted Extensions インタフェースとともにこのマニュアルで使用されます。

- `int auditon(int cmd, caddr_t data, int length);`
- `void free(void *ptr);`
- `int getpeerucred(int fd, ucred_t **ucred);`
- `uint_t getpflags(uint_t flag);`
- `int is_system_labeled(void);`
- `int setpflags(uint_t flag, uint_t value);`
- `int getsockopt(int s, int level, int optname, void *optval, int *optlen);`
- `int setsockopt(int s, int level, int optname, const void *optval, int optlen);`
- `int socket(int domain, int type, int protocol);`
- `ucred_t *ucred_get(pid_t pid);`
- `m_label_t *ucred_getlabel(const ucred_t *uc);`

Trusted Extensions のシステムコールとライブラリルーチン

次の表に Trusted Extensions システムコールとルーチンを示します。さらに、インタフェースの説明と宣言についてのこのマニュアルにおける参照先、およびインタフェースの例の参照先を示します。マニュアルページセクションは、各システムコールおよびルーチンの名前の一部として含まれます。

表 B–1 Trusted Extensions で使用されるシステムコールとライブラリルーチン

システムコールまたはライブラリルーチン	説明の相互参照	例の相互参照
`bldominates(3TSOL)`	17 ページの「ラベル関係」 41 ページの「ラベルの比較」	50 ページの「2 つのラベル間の関係の判別」
`blequal(3TSOL)`	41 ページの「ラベルの比較」	50 ページの「2 つのラベル間の関係の判別」
`blinrange(3TSOL)`	17 ページの「ラベル関係」	61 ページの「プリンタのラベル範囲に対するラベル要求の検査」
`blmaximum(3TSOL)`	41 ページの「ラベルの比較」	
`blminimum(3TSOL)`	41 ページの「ラベルの比較」	

表 B–1 Trusted Extensions で使用されるシステムコールとライブラリルーチン (続き)

システムコールまたはライブラリルーチン	説明の相互参照	例の相互参照
`blstrictdom(3TSOL)`	41 ページの「ラベルの比較」	
`fgetlabel(2)`	29 ページの「ラベル付けされたゾーン」 36 ページの「ファイルのラベルの取得と設定」	
`free(3C)`	39 ページの「ラベルと文字列との変換」	
`getdevicerange(3TSOL)`	37 ページの「ラベル範囲の取得」	61 ページの「プリンタのラベル範囲に対するラベル要求の検査」
`getlabel(2)`	29 ページの「ラベル付けされたゾーン」 36 ページの「ファイルのラベルの取得と設定」	48 ページの「ファイルラベルの取得」
`getpathbylabel(3TSOL)`	38 ページの「ゾーンのラベルへのアクセス」	
`getpeerucred(3C)`	57 ページの「`get_peer_label()` ラベル対応関数」	59 ページの「資格と遠隔ホストラベルの取得」
`getpflags(2)`	26 ページの「MAC 適用外ソケット」	
`getplabel(3TSOL)`	35 ページの「プロセス機密ラベルへのアクセス」	86 ページの「フォントリストによるウィンドウラベルの変換」
`getuserrange(3TSOL)`	37 ページの「ラベル範囲の取得」	
`getzoneidbylabel(3TSOL)`	38 ページの「ゾーンのラベルへのアクセス」	
`getzonelabelbyid(3TSOL)`	38 ページの「ゾーンのラベルへのアクセス」	
`getzonelabelbyname(3TSOL)`	38 ページの「ゾーンのラベルへのアクセス」	
`getzonerootbyid(3TSOL)`	38 ページの「ゾーンのラベルへのアクセス」	
`getzonerootbylabel(3TSOL)`	38 ページの「ゾーンのラベルへのアクセス」	
`getzonerootbyname(3TSOL)`	38 ページの「ゾーンのラベルへのアクセス」	
`is_system_labeled(3C)`	57 ページの「`get_peer_label()` ラベル対応関数」 34 ページの「Trusted Extensions システムの検出」	58 ページの「ラベル付けされている環境で印刷サービスが実行されているか否かの判別」

表 B–1 Trusted Extensions で使用されるシステムコールとライブラリルーチン (続き)

システムコールまたはライブラリルーチン	説明の相互参照	例の相互参照
labelbuilder(3TSOL)	第 7 章	90 ページの「対話型ユーザーインタフェースの作成」
label_to_str(3TSOL)	39 ページの「ラベルと文字列との変換」	47 ページの「プロセスラベルの取得」
m_label_alloc(3TSOL)	35 ページの「ラベルのためのメモリーの割り当てと解放」	47 ページの「プロセスラベルの取得」
		48 ページの「ファイルラベルの取得」
m_label_dup(3TSOL)	35 ページの「ラベルのためのメモリーの割り当てと解放」	
m_label_free(3TSOL)	35 ページの「ラベルのためのメモリーの割り当てと解放」	61 ページの「プリンタのラベル範囲に対するラベル要求の検査」
		47 ページの「プロセスラベルの取得」
setflabel(3TSOL)	36 ページの「ファイルのラベルの取得と設定」	
	36 ページの「ファイルのラベルの取得と設定」	
setpflags(2)	26 ページの「MAC 適用外ソケット」	
str_to_label(3TSOL)	39 ページの「ラベルと文字列との変換」	61 ページの「プリンタのラベル範囲に対するラベル要求の検査」
		48 ページの「ファイルラベルの取得」
tsol_getrhtype(3TSOL)	39 ページの「遠隔ホストタイプの取得」	
ucred_get(3C)	26 ページの「マルチレベルポート」	
ucred_getlabel(3C)	26 ページの「マルチレベルポート」	
XTSOLgetClientAttributes(3XTSOL)	80 ページの「属性へのアクセス」	
XTSOLgetPropAttributes(3XTSOL)	80 ページの「属性へのアクセス」	
XTSOLgetPropLabel(3XTSOL)	82 ページの「ウィンドウプロパティーラベルへのアクセスと設定」	
XTSOLgetPropUID(3XTSOL)	82 ページの「ウィンドウプロパティーラベルへのアクセスと設定」	
XTSOLgetResAttributes(3XTSOL)	85 ページの「ウィンドウ属性の取得」	

表 B–1 Trusted Extensions で使用されるシステムコールとライブラリルーチン (続き)

システムコールまたはライブラリルーチン	説明の相互参照	例の相互参照
XTSOLgetResLabel(3XTSOL)	87 ページの「ウィンドウラベルの取得」	
XTSOLgetResUID(3XTSOL)	87 ページの「ウィンドウユーザー ID の取得」	
	81 ページの「ウィンドウユーザー ID へのアクセスと設定」	
XTSOLgetSSHeight(3XTSOL)	84 ページの「スクリーンストライプの高さへのアクセスと設定」	
XTSOLgetWorkstationOwner(3XTSOL)	82 ページの「ワークステーション所有者 ID へのアクセスと設定」	
XTSOLIsWindowTrusted(3XTSOL)	83 ページの「トラステッドパスウィンドウでの作業」	
XTSOLMakeTPWindow(3XTSOL)	83 ページの「トラステッドパスウィンドウでの作業」	
XTSOLsetPolyInstInfo(3XTSOL)	第 6 章	
XTSOLsetPropLabel(3XTSOL)	82 ページの「ウィンドウプロパティーラベルへのアクセスと設定」	
XTSOLsetPropUID(3XTSOL)	82 ページの「ウィンドウプロパティーラベルへのアクセスと設定」	
XTSOLsetResLabel(3XTSOL)	87 ページの「ウィンドウラベルの設定」	
XTSOLsetResUID(3XTSOL)	81 ページの「ウィンドウユーザー ID へのアクセスと設定」	
XTSOLsetSessionHI(3XTSOL)	83 ページの「X ウィンドウサーバーの認可上限と最下位ラベルの設定」	
XTSOLsetSessionLO(3XTSOL)	83 ページの「X ウィンドウサーバーの認可上限と最下位ラベルの設定」	
XTSOLsetSSHeight(3XTSOL)	84 ページの「スクリーンストライプの高さへのアクセスと設定」	
XTSOLsetWorkstationOwner(3XTSOL)	82 ページの「ワークステーション所有者 ID へのアクセスと設定」	

索引

F

G

I

J

S

T

U

W

X

れ