Trusted Extensions 開発者ガイド

Part No: E38869–02
2013 年 3 月

ORACLE

130701@25097

目次

はじめに

Trusted Extensions 開発者ガイドでは、アプリケーションプログラミングインタフェース (API) を使用して、Oracle Solaris OS の Trusted Extensions 機能で構成される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方法について説明します。読者は、UNIX プログラミングに精通し、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 – この Oracle Solaris 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を使用するシステムをサポートしています。サポートされるシステムは、Oracle Solaris OS: Hardware Compatibility Lists に記載されています。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 に関連する用語については、次を参照してください。

- x86 は、64 ビットおよび 32 ビットの x86 互換製品系列を指します。
- x64 は特に 64 ビット x86 互換 CPU 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、Oracle Solaris OS: Hardware Compatibility Listsを参照してください。

このドキュメントに記載されているプログラム例は、API の紹介を中心としており、エラーチェックを行なっていないことに注意してください。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行うようにしてください。

Trusted Extensions のドキュメントの構成

Trusted Extensions のドキュメントセットは、Oracle Solaris リリースのドキュメントを補足します。Trusted Extensions をさらに詳しく理解するには、両方のドキュメントセットを参照してください。Trusted Extensions のドキュメントセットは、次のドキュメントで構成されています。

ドキュメントのタイトル	内容	対象読者
『Solaris Trusted Extensions 移行ガイド』	廃止。Trusted Solaris 8 ソフトウェア、Oracle Solaris ソフトウェア、および Trusted Extensions ソフトウェア間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 Oracle Solaris OS の『新機能』のドキュメントで概説しています。	すべてのユーザー
『Solaris Trusted Extensions リファレンスマニュアル』	廃止。Solaris 10 11/06 と Solaris 10 8/07 リリースの Trusted Extensions における 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 Oracle 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
『Trusted Extensions User's Guide 』	Trusted Extensions. の基本的な機能について説明しています。用語集も付属しています。	エンドユーザー、管理者、開発者
『Oracle Solaris Trusted Extensions 構成ガイド』	Solaris 10 5/08 リリース以降において、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。『Solaris Trusted Extensions インストールと構成』に代わるものです。	管理者、開発者
『Trusted Extensions 管理者の手順』	具体的な管理タスクの実行方法を示します。	管理者、開発者
『Trusted Extensions 開発者ガイド』	Trusted Extensions を使ってアプリケーションを開発する方法について説明しています。	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でのラベルコンポーネントの指定方法について説明します。	管理者
『Compartmented Mode Workstation Labeling: Encodings Format 』	ラベルエンコーディングファイルで使用される構文について説明します。構文を使用することにより、適格な形式のラベルに関するさまざまな規則がシステムに適用されます。	管理者

内容の紹介

第 1 章「Trusted Extensions API およびセキュリティーポリシー」では、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「ラベルと認可上限」では、プロセスおよびデバイスオブジェクトの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、この章では、認可上限、プロセスで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「ラベルのコード例」では、ラベル用の API を使用するコード例を示します。

第 4 章「印刷とラベル API」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「プロセス間通信」では、同じワークステーション内やネットワーク経由でのプロセス間通信に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「Trusted X Window System」では、管理アプリケーションによるセキュリティー関連の X Window System 情報へのアクセスと変更を可能にするデータ型および API について説明します。この章にはコード例のセクションもあります。

第 7 章「ラベルビルダー API」では、ラベルと認可上限の生成に使用する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。この章にはコード例のセクションもあります。

第 8 章「信頼できる Web ガードプロトタイプ」では、インターネットを通じた攻撃から Web サーバーとその Web コンテンツを隔離する、安全な Web ブラウジングプロトタイプの例を示します。

第 9 章「Solaris Trusted Extensions ラベル API のための試験的な Java バインディング」では、Trusted Extensions ソフトウェアに付属しているラベル API をミラー化する、実験的な一連の Java クラスおよびメソッドについて説明します。この章にはソースコードへのポインタや構築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「プログラマーのリファレンス」では、Trusted Extensions のマニュアルページ、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。この付録では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「Trusted Extensions API リファレンス」では、パラメータおよび戻り値の宣言など、プログラミングインタフェースの一覧を示します。

Oracle サポートへのアクセス

Oracle のお客様は、My Oracle Support を通じて電子的なサポートを利用することができます。詳細は、`http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` を参照してください。聴覚に障害をお持ちの場合は、`http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` を参照してください。

表記上の規則

次の表では、このドキュメントで使用される表記上の規則について説明します。

表 P–1 表記上の規則

字体	説明	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。 `machine_name% you have mail.`
`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。	`machine_name%` **`su`** `Password:`
aabbcc123	プレースホルダ: 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。
AaBbCc123	書名、新しい単語、および強調する単語を示します。	『ユーザーズガイド』の第 6 章を参照してください。 キャッシュは、ローカルに格納されるコピーです。 ファイルを保存しないでください。 注: いくつかの強調された項目は、オンラインでは太字で表示されます。

コマンド例のシェルプロンプト

Oracle Solaris OS に含まれるシェルで使用する、UNIX のシステムプロンプトとスーパーユーザープロンプトを次に示します。コマンド例のシェルプロンプトから、通常ユーザーと特権ユーザーのどちらがコマンドを実行すべきかがわかります。

表 P–2 シェルプロンプト

シェル	プロンプト
Bash シェル、Korn シェル、および Bourne シェル	$
Bash シェル、Korn シェル、および Bourne シェルのスーパーユーザー	#

表 P–2 シェルプロンプト (続き)

シェル	プロンプト
C シェル	machine_name%
C シェルのスーパーユーザー	machine_name#

◆ ◆ ◆ 第 1 章

Trusted Extensions API およびセキュリティーポリシー

Oracle Solaris OS の Trusted Extensions 機能 (Trusted Extensions) は、ラベルへのアクセスおよび処理を行うアプリケーションを作成できるアプリケーションプログラミングインタフェース (API) を提供します。この章では API 機能の概要を示し、Trusted Extensions のセキュリティーポリシーについて紹介します。

Trusted Extensions の用語の定義については、『Trusted Extensions User's Guide』の用語集を参照してください。

この章で扱う内容は、次のとおりです。

ラベルの理解

Trusted Extensions ソフトウェアでは、Oracle Solaris OS のセキュリティー機能を拡張するポリシーおよびサービスのセットが提供されています。これらの拡張は、ラベルの関係に基づくアクセス制御を提供します。

ラベルはデータへのアクセスを制御し、データの格付けを保持します。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」とは、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適用される規則のセットです。「セキュリティーポリシー」という用語は、ポリシー自体または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。

このセクションには、ラベルのタイプ、範囲、コンポーネント、および関係についての概要が記載されています。

ラベルのタイプ

Trusted Extensions ソフトウェアでは、機密ラベルと認可上限ラベルという 2 つのタイプのラベルが定義されています。「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、通常は「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、通常は「認可上限」と呼ばれます。

機密ラベル

Trusted Extensions ソフトウェアで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で格納します。それぞれのレベルは、ある機密ラベルを持つ固有のゾーンに関連付けられます。機密ラベルは、そのゾーン内の情報の機密性を指定し、その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。ラベルは CONFIDENTIAL、SECRET、あるいは TOP SECRET のようになります。「サブジェクト」とは、プロセスなどのアクティブなエンティティーで、オブジェクト間で情報を移動させたり、システムの状態を変更したりします。「オブジェクト」とは、ファイルやデバイスなど、データを保持したり受け取ったりする受動的なエンティティーです。あるゾーンで実行するすべてのプロセスや、ゾーンに格納されているすべてのファイルなどは、それら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) による決定で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に表示されます。

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。認可上限とは、ラベル範囲の上限を定義するラベルです。たとえば、認可上限が SECRET の場合、こ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、これより高い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してプロセスを開始できる、最上位のラベルです。言い換えると、ユーザー認可上限とは、ユーザーのアカウントラベル範囲の上限です。ユーザーはログインすると、自分のセッション認可上限を選択します。「セッション認可上限」は、ユーザーがアクセスできるラベルを決定します。セッション認可上限は、ユーザーがログインセッション中にファイルにアクセスしてプロセスを開始できる、「最低の上限」を設定します。セッション認可上限よりもユーザー認可上限の方が優位になります。

ラベル範囲

セキュリティー管理者は、ラベルの範囲とラベルセットを定義して、「必須アクセス制御」 (MAC) ポリシーを適用します。「ラベル範囲」は、認可上限または制限値による上端と、最小ラベルによる下端によって境界を設定されたラベルのセットで

す。「ラベル制限」は、ラベル範囲の上限です。「ラベルセット」には、1 つ以上の個別ラベルが含まれ、これらは互いに無関係であることもあります。ラベルセット内のラベル間に優位性はありません。

ラベルコンポーネント

ラベルには、階層化された格付けと、階層化されていないゼロ個以上のコンパートメントのセットが含まれています。格付けは「レベル」あるいはセキュリティーレベルとも呼ばれます。「格付け」はラベル階層内の単一レベルを表し、`TOP SECRET` や `UNCLASSIFIED` などがあります。「コンパートメント」は格付けに関連付けられ、人事 (HR) グループまたは販売グループの秘密情報などの、システム内の階層化されていない個別の情報領域を表します。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` の格付けを持つユーザーは、関連付けられたコンパートメントのリストによって指定される秘密情報にのみアクセスでき、ほかの秘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を組み合わせて、ゾーンとそのゾーン内のリソースのラベルを表します。

格付けのテキスト形式は、`label_encodings` ファイルに指定され、次のように表示されます。

```
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;
```

コンパートメントのテキスト形式は、`label_encodings` ファイルに指定され、次のように表示されます。

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

ラベルの定義およびラベル形式の詳細については、『Solaris Trusted Extensions ラベルの管理』および『Compartmented Mode Workstation Labeling: Encodings Format 』を参照してください。ラベルの API については、第 2 章「ラベルと認可上限」を参照してください。

ラベルの関係

ラベルの比較は、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を意味し、ターゲットのラベルは機密ラベルあるいは認可上限ラベルである場合があります。比較の結果に基づいて、プロセスに対してオブジェクトへのアクセス権が付与されるか、アクセスが拒否されます。アクセス権が付与されるのは、プロセスのラベルがターゲットのラベルよりも優位な場合に限られます。ラベルの関係と優位性については、このセクションで後述します。この例については、48 ページの「2 つのラベル間の関係の特定」を参照してください。

「セキュリティーレベル」は数値による格付けです。ラベルはエンティティーのセキュリティーレベルを示し、ゼロ以上のコンパートメントが含まれている場合があります。エンティティーは、プロセス、ゾーン、ファイル、デバイスなどのラベル付け可能な対象です。

ラベルには次のタイプがあり、次のように相互に関連しています。

- 同等 – 1 つのラベルが別のラベルと同等のとき、次の両方の文が真になります。
  - ラベルの格付けが別のラベルの格付けと数値的に同等である。
  - ラベルは別のラベルとまったく同じコンパートメントを持つ。
- 優位 – 1 つのラベルが別のラベルよりも優位のとき、次の両方の文が真になります。
  - ラベルの格付けが別のラベルの格付けよりも数値的に大きいか同等である。
  - ラベルは別のラベルとまったく同じコンパートメントを持つ。
- 完全に優位 – 1 つのラベルが別のラベルよりも完全に優位のとき、次の両方の文が真になります。
  - ラベルの格付けが別のラベルの格付けよりも数値的に大きいか同等である。
  - ラベルは別のラベルが持つすべてのコンパートメントを持ち、それ以外のコンパートメントを少なくとも 1 つ持つ。
- 無関係 – 1 つのラベルと別のラベルが無関係であるとき、次の両方の文が真になります。
  - ラベルが同等でない。
  - いずれのラベルも他方のラベルより優位ではない。

ラベルの格付けとコンパートメントを指定するには `label_encodings` ファイルを使用します。`label_encodings(4)` のマニュアルページを参照してください。

いずれかのタイプのラベルが持つセキュリティーレベルが、別のラベルのセキュリティーレベルと同等かそれより高い場合、最初のラベルは 2 番目のラベルよりも「優位」であると言います。このセキュリティーレベルの比較は、ラベルの格付けおよびコンパートメントに基づきます。優位なラベルの格付けは、2 番目のラベルの格付けと等しいか、それより高いことが必要です。また、優位なラベルには、2 番目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 つの同等のラベルは、互いに優位であると言えます。

`label_encodings` ファイルから抜粋した次の例で、`REGISTERED (REG)` ラベルは `CONFIDENTIAL (C)` ラベルより優位です。この比較は各ラベルの `value` キーワードの値に基づいています。`REG` ラベルの `value` キーワードの値は、`C` ラベルの `value` キーワードの値以上になっています。どちらのラベルも `PUBLIC (P)` ラベルよりも優位です。

initial compartments キーワードの値は、格付けに最初に関連付けられているコンパートメントのリストを示します。initial compartments キーワード内のそれぞれの数値は「コンパートメントビット」で、それらはすべて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;
```

次に示す label_encodings からの抜粋では、REG HR ラベル (人事) が REG ラベルよりも優位であることを示しています。REG HR ラベルは REGISTERED の格付けと、HR のコンパートメントを持ちます。HR コンパートメントの compartments キーワードには 0 コンパートメントビットが設定されているため、REG HR の格付けには、REG の格付けによって設定されるコンパートメントよりも多い、コンパートメント 0、4–5、および 190–239 が設定されます。

```
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、完全に優位であることが求められる場合があります。前の例で、REG ラベルは P ラベルよりも完全に優位で、REG HR ラベルは REG ラベルよりも完全に優位です。ラベルを比較するとき、ある REG ラベルは別の REG ラベルよりも優位です。

互いに優位でないラベルは、無関係と呼ばれます。「無関係」のラベルは企業内の部門を分離するとき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) と REG Sales ラベルが無関係であることが定義されています。これらのラベルが無関係であるのは、各コンパートメントに設定されているコンパートメントビットが異なるためです。

```
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

このセクションでは、本ドキュメントで説明する次の Trusted Extensions API について紹介します。

- ラベル API
- Trusted X Window System API
- ラベルビルダー API

これらの Trusted Extensions API に加えて、Oracle Solaris OS で使用可能なセキュリティー API も使用することができます。Trusted Extensions 上で動作するアプリケーションによって、別のセキュリティー属性を操作することが必要になる場合があります。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、許可、およびプロファイルについての情報が格納されています。これらのデータベースは、プログラムを実行できるユーザーを制限できます。特権はさまざまな Oracle Solaris プログラムにコーディングされており、サードパーティーアプリケーションにもコーディングすることができます。

これらの Oracle Solaris OS セキュリティー API の詳細については、『Oracle Solaris 10 セキュリティー開発者ガイド』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。

Oracle Solaris OS で提供されている「任意アクセス制御」 (DAC) では、データの所有者が、データへのアクセスが許可されるユーザーを決定します。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が提供されています。MAC では、通常のユーザーが「セキュリティーポリシー」を指定したりオーバーライドしたりすることはできません。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、ユーザー、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で許可される場合、API によって、ユーザープロセスまたは役割プロセスにラベルを設定することができます。ゾーンまたはホストへのラベルの設定は管理上の手順であり、プログラムによる手順ではありません。

ウィンドウラベルをカスタマイズするアプリケーションを作成できます。Trusted Extensions ソフトウェアには、基本的なラベル作成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 ベースのプログラムインタフェースが提供されています。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有効な認可上限を対話的に作成できます。

ラベル API は隠されたラベル上で動作します。「隠されたラベル」では、ラベルの内部構造は公開されません。隠されたラベルを使用することによって、API を使用して作成された既存のプログラムは、ラベルの内部構造が変更した場合でも機能できます。たとえば、ラベル API を使用してラベル内の特定のビットを指定することは

できません。ラベル API によって、ラベルを取得したりラベルを設定したりできます。ラベルを設定できるのは、セキュリティーポリシーによって許可されている場合に限られます。

ラベル API

ラベル、ラベル範囲、およびラベル制限によって、Trusted Extensions で構成されたシステム上の情報にアクセスできるユーザーが決定されます。

ラベル API は、ラベル、ラベルの範囲と制限、およびラベル間の関係にアクセスしたり、これらを変換したり、比較を実行したりするために使用されます。あるラベルが別のラベルより優位であったり、ほかのラベルと無関係あったりする場合もあります。

`label_encodings` ファイルには、Trusted Extensions 環境に関する機密ラベル、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外観も制御します。セキュリティー管理者は `label_encodings` ファイルを作成および保守する役割を担っています。`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ているか、場合によってはラベル範囲に関連付けられています。オブジェクトには、定義されたラベル範囲内の特定のラベルでアクセスできます。ラベル範囲に関連付けられているオブジェクトには、次のものがあります。

- すべてのユーザーおよびすべての役割
- 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
- テープドライブ、ディスクドライブ、CD-ROM デバイス、およびオーディオデバイスなどの割り当て可能なデバイス
- プリンタやワークステーションなどのその他の割り当て不能なデバイス

  ワークステーションのアクセスは、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設定されてい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、デバイスマネージャー GUI を使用してこの範囲を設定します。デフォルトでは、デバイスは `ADMIN_LOW` から `ADMIN_HIGH` の範囲を持ちます。

ラベルの詳細については、16 ページの「ラベルのタイプ」を参照してください。

アクセス制御の決定にラベルが使用される方法

MAC はアプリケーションを実行しているプロセスのラベルと、プロセスがアクセスしようとしているオブジェクトのラベルまたはラベル範囲とを比較します。MAC は低いラベルまでの読み取りをプロセスに許可し、同等のラベルへの書き込みをプロセスに許可します。

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

マルチレベルポート (MLP) にバインドされたプロセスは、複数のラベルでリクエストを待機でき、リクエストの発信元に応答を送信できます。Trusted Extensions では、このような応答は同位書き込みです。

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

ラベル API のタイプ

機密ラベル API

機密ラベル API を使用して、次のことを実行できます。

- プロセスラベルの取得
- ラベルの初期化
- 2 つのラベル間での最大の下限または最小の上限の検索
- ラベルの優位性および同等性の比較
- ラベルタイプの確認および設定
- 読み取り可能な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ってシステム範囲内にあることの確認

これらの API の説明については、第 2 章「ラベルと認可上限」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。範囲の上限が事実上の認可上限です。範囲の上限と下限が等しい場合、範囲は単一のラベルです。

認可上限ラベル API を使用して、次のことを実行できます。

- 2 つのラベル間での最大の下限または最小の上限の検索
- ラベルの優位性および同等性の比較
- 内部形式と 16 進形式の間での認可上限の変換

これらの API の説明については、第 2 章「ラベルと認可上限」を参照してください。

ラベル範囲 API

ラベル範囲を使用して、次のラベルに制限を設定できます。

- ホストが情報を送信および受信できるラベル
- ユーザーおよび役割に代わって機能しているプロセスが、システム上で動作できるラベル
- ユーザーがデバイスを割り当てることができるラベル

  このラベル範囲の使用により、これらのデバイス上のストレージメディアにファイルを書き込む際のラベルを制限できます。

ラベル範囲は管理者によって割り当てられます。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。

ラベル範囲に関する情報を取得するには、次の方法を使用できます。

- `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。

これらの API の説明については、第 2 章「ラベルと認可上限」を参照してください。

Trusted X Window System API

Trusted X Window System バージョン 11 サーバーは、ログイン時に起動します。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、選択、および ToolTalk セッションが、別々の個別オブジェクトとして複数の機密ラベルで作成されます。複数の機密ラベルでの個別オブジェクトの作成は、「ポリインスタンス化」と呼ばれます。Motif ウィジェット、Xt Intrinsics、Xlib、およびデスクトップインタフェースで作成されたアプリケーションは、セキュリティーポリシーの制約の範囲内で実行します。これらの制約は、X11 プロトコルの拡張によって強制されます。

第 6 章「Trusted X Window System」では、24 ページの「Trusted Extensions セキュリティーポリシー」に記載されているセキュリティー属性情報にアクセスできるプログラムインタフェースが記載されています。これらのプログラムインタフェースは、ラベルおよび認可上限をテキストに変換するためにも使用できます。テキストは、Trusted X Window System で表示する幅およびフォントリストを指定することによって制約を加えることができます。

Trusted X Window System は次のセキュリティー属性を格納します。

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

トラステッドパスフラグは、ウィンドウをトラステッドパスウィンドウとして識別します。トラステッドパスウィンドウは、信頼されていないプログラムによるアクセスからシステムを保護します。このウィンドウは、スクリーンストライプまたはログインウィンドウなどの常に最前面のウィンドウです。

付録 B 「Trusted Extensions API リファレンス」に、信頼できる X11 IPC パスを作成するために使用できる拡張の一覧が示されています。

ラベルビルダー API

Trusted Extensions ソフトウェアには、アプリケーションのグラフィカルユーザーインタフェース (GUI) をユーザーが作成できる、ラベルビルダー API が提供されています。GUI によってユーザー入力を受け取り、その入力から有効なラベルを作成します。

Trusted Extensions で構成されたシステムでは、基本的なラベル作成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 ベースのプログラムインタフェースが提供されています。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有効な認可上限を対話的に作成できます。これらのプログラムインタフェースについては、第 7 章「ラベルビルダー API」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、データの格付けを保持します。すべてのプロセスおよびオブジェクトは、MAC による決定で使用される機密ラベルを持っています。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」とは、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規則のセットです。

次のセクションでは、Trusted Extensions のセキュリティーポリシーがマルチレベル動作、ゾーン、およびラベルにどのように影響するかについて説明します。

マルチレベル操作

複数のセキュリティーレベルで実行する操作を作成するとき、次の問題について考慮する必要があります。

- 大域ゾーン内のライトダウンポリシー
- 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 除外ソケット

大域ゾーン内のプロセスのみが、指定されたラベルでプロセスを開始できるため、複数のセキュリティー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。

大域ゾーン内のライトダウンポリシー

プロセスなどのサブジェクトが、そのサブジェクトより優位性の低いラベルを持つオブジェクトを書き込む能力を「ライトダウン」と言います。大域ゾーンのライトダウンポリシーは管理的に指定されます。大域ゾーンのプロセスは `ADMIN_HIGH` ラベルで実行するため、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を大域ゾーン内に読み取り/書き込みでマウントすることができます。ただし、これらの特殊なファイルシステムマウントをオートマウントマップ内で管理的に指定する必要があり、これらは大域ゾーンオートマウンタによってマウントする必要があります。これらのマウントでは、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の内部にマウントポイントを持つ必要があります。ただし、これらのマウントポイントは、ラベル付けされたゾーンの内部から可視であってはいけません。

たとえば、`PUBLIC` ゾーンにゾーンパス `/zone/public` がある場合、`/zone/public/home/mydir` という書き込み可能なマウントポイントは許可されます。ただし、`/zone/public/root/home/mydir` という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ますが、大域ゾーンによってアクセスできないため、許可されません。ゾーンを超えた NFS マウントは許可されません。つまり、NFS マウントされたファイルは、ファイルシステムをマウントしたゾーン内で実行するプロセスによってのみアクセスできるということを意味します。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、このようなファイルにライトダウンすることができます。

ゾーンに関連付けられたローカルファイルシステムは、ファイルの「アクセス権」および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からのアクセスから保護されます。各ゾーンの root (/) ディレクトリの親ディレクトリは、`root` プロセスか、`file_dac_search` 特権を表明するプロセスによってのみアクセス可能です。

一般に、大域ゾーンからライトダウンする機能は制限されています。通常、ライトダウンが使用されるのは、`setflabel()` インタフェースを使用してファイルがふたたび格付けされる場合や、特権ユーザーがファイルマネージャーアプリケーションの別のゾーンにファイルをドラッグアンドドロップする場合に限られます。

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。この属性は、ネットワークデータベースファイル内の設定に従って割り当てられます。ホストタイプ、それらで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については、『Trusted Extensions 管理者の手順』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

データを送信または受信するネットワーク操作のデフォルトポリシーでは、ローカルプロセスとリモートピアが同じラベルを持つ必要があります。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めたすべてのゾーンに適用されます。ただし、デフォルトのネットワークポリシーは、ファイルシステムのマウント用のポリシーよりも柔軟性があります。Trusted Extensions には、デフォルトのネットワークポリシーをオーバーライドするための管理インタフェースとプログラムインタフェースが提供されています。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場合は、十分に注意してください。このメカニズムを使用する必要がある場合、使用しているサーバーアプリケーションが MAC ポリシーを適用し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一覧で示されます。ゾーン内部のプロセスが `net_bindmlp` 特権を表明する場合、これらのプロセスは 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` 特権も表明される必要があります。このようなバインディングにより、プロセスがバインドされている IP アドレスに関連付けられたすべてのラベルで、プロセスが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たラベルは、`tnrhdb` データベースと `tnrhtp` データベースで指定されます。ラベルの指定は、範囲、明示的に列挙されたラベルのセット、または両方の組み合わせによって行うことができます。

MLP にバインドされた特権付きプロセスが TCP リクエストを受け取ると、応答はリクエスト側のラベルを付けて自動的に送信されます。UDP データグラムの場合、SO_RECVUCRED オプションで指定されたラベルを付けて応答が送信されます。

特権付きプロセスには、リクエストのラベルをほかのパラメータと比較することによって、さらに制限された MAC ポリシーを実装することができます。たとえば、Web サーバーは、リクエスト側プロセスのラベルと URL で指定されたファイルのラベルとを比較できます。リモートラベルは、リモートピアの資格情報を返す getpeerucred() 関数を使用して判別することができます。ピアが同じホスト上のゾーン内で実行中の場合、ucred_get() ライブラリルーチンによって資格情報の完全なセットが返されます。ピアがローカルであるかリモートであるかに関係なく、ピアのラベルは ucred_getlabel() 関数を使用することによって ucred データ構造からアクセスできます。このラベルは bldominates() などの関数を使用することによって、ほかのラベルと比較できます。

ゾーンはシングルレベルポートおよびマルチレベルポートを持つことができます。63 ページの「マルチレベルポート情報」を参照してください。

MAC 除外ソケット

Trusted Extensions ソフトウェアには、エンドポイントと低いラベルで通信するためにソケットを使用できることを指定する、明示的なソケットオプション SO_MAC_EXEMPT が提供されています。

注意 – SO_MAC_EXEMPT ソケットオプションは不用意に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ときは、十分に注意してください。このメカニズムを使用する必要がある場合、使用しているクライアントアプリケーションが MAC ポリシーを適用していることを確認してください。

Trusted Extensions ソフトウェアでは、SO_MAC_EXEMPT オプションの使用が次のように制限されています。

- ソケットオプションを明示的に設定するには、プロセスが 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、通常ユーザーの制限セットから net_mac_aware 特権を削除することができます。

詳細は、user_attr(4) のマニュアルページを参照してください。

場合によっては、ソケットがライブラリによって管理されるときなど、ソケットオプションを明示的に設定することが実用的でないこともあります。このような場合、ソケットオプションを暗黙的に設定することができます。setpflags() システムコールによって、ユーザーは NET_MAC_AWARE 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_aware 特権も必要です。プロセスフラグが有

効になっている間に開いているすべてのソケットには、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_aware 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、アプリケーションによって開かれたすべてのソケットには、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は、このプロセスフラグまたは関連した特権を持ちません。

SO_MAC_EXEMPT ソケットオプションを使用する必要があるときは、可能な場合は常に、アプリケーションのソースコードを精査して変更してください。このようなコードへの変更ができない場合や安全な方法が使用できない場合、ppriv -M コマンドを使用してもかまいません。

SO_MAC_EXEMPT ソケットオプションは Oracle Solaris OS ではあまり使用されていませんでした。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、信頼されないオペレーティングシステム上で異なるラベルで実行している NFS サーバーと通信することが必要な場合があります。NFS クライアントは MAC ポリシーを適用して、不適切なリクエストが認められないようにします。

Oracle Solaris OS では、NFS サーバーおよびクライアントコードの両方に MAC ポリシーが組み込まれて MAC ポリシーを適用することで、Oracle Solaris クライアントまたはサーバーと、信頼されないクライアントまたはサーバーの間の通信で MAC ポリシーが使用可能になります。信頼されないホストが Trusted Extensions を実行するシステムと通信できるようにするには、信頼されないホストが tnrhdb データベースにエントリを持つ必要があります。詳細は、『Trusted Extensions 管理者の手順』の「トラステッドネットワークデータベースの構成 (タスクマップ)」を参照してください。

ゾーンとラベル

Trusted Extensions で構成されたシステム上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。このようなゾーンは「ラベル付きゾーン」と呼ばれます。ラベル付きゾーンは非大域ゾーンで、通常のユーザーからアクセスできます。複数のラベルでクリアされたユーザーは、それらの各レベルでゾーンへのアクセスが許可されます。

「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が格納されている特殊なゾーンです。大域ゾーン内のファイルには、役割と特権プロセスによってのみアクセスできます。

大域ゾーン内のラベル

大域ゾーンにはラベルの範囲が割り当てられます。範囲は ADMIN_LOW から ADMIN_HIGH まであります。ADMIN_HIGH および 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内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 ディレクトリ内のファイルには、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、パッケージ化またはパッチ適用の手順中を除けば一般的には変更されません。ADMIN_LOW ファイルを変更するには、通常の場合、スーパーユーザーまたはすべての特権を持つユーザーが、プロセスを実行する必要があります。

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、大域ゾーン内のすべてのプロセスと、/etc ディレクトリ内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。25 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、大域ゾーン内でプロセスを実行できます。

ラベル付きゾーン

非大域ゾーンは「ラベル付きゾーン」と呼ばれます。各ラベル付きゾーンには一意のラベルがあります。ラベル付きゾーン内のすべてのオブジェクトには、同じラベルがあります。たとえば、ラベル付きゾーン内で実行するすべてのプロセスは、同じラベルを持ちます。ラベル付きゾーン内で書き込み可能なすべてのファイルは、同じラベルを持ちます。複数のラベルについてクリアされたユーザーは、各レベルでラベル付きゾーンにアクセスできます。

Trusted Extensions では、ゾーンのためのラベル API のセットが定義されています。これらの API は、ラベル付きゾーンに関連付けられたラベルと、それらの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 の詳細については、37 ページの「ゾーン内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いています。したがって、ファイルのラベルを変更する場合、ファイルは適切なラベル付きゾーンまたは適切なラベル付きホストに移動する必要があります。ファイルのラベルを変更するこのプロセスは、ファイルの「再格付け」とも言

います。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ラベルを変更することができます。ファイルのラベルを変更するには、プロセスは 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。

特権の設定についての詳細は、『Oracle Solaris 10 セキュリティー開発者ガイド』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、ラベルと認可上限の比較など、基本的なラベル操作を実行するための Trusted Extensions の API について説明します。また、この章では、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。

この章で扱う内容は、次のとおりです。

- 31 ページの「特権操作とラベル」
- 33 ページの「ラベル API」
- 42 ページの「機密ラベルの取得」

第 3 章「ラベルのコード例」には、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ド例が記載されています。

特権操作とラベル

操作でセキュリティーポリシーを省略またはオーバーライドできるようにするには、その操作の実効セットに特別な特権を含める必要があります。

実効セットへの特権の追加は、プログラム上で、または管理者によって、このように行われます。

- 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、それは実効セットにすべての特権が含まれた状態で開始されます。たとえば、CDE ファイルマネージャーはその実効セットにすべて特権が含まれた状態で開始されます。その後、ファイルマネージャーはその特権のほとんど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ものだけを保持します。
- 管理者は、SMF サービス用のマニフェストファイル内、または一般コマンド用の RBAC データベース `exec_attr` ファイル内に特権を指定できます。このファイルの詳細は、`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、特別な特権を使って操作を実行できます。これらの特権は、「権利プロファイル」を使って指定できます。同様に、特定の特権を使って特定の関数を実行す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は、必ず特権を必要とする関数を実行している間のみその特権を有効にし、関数が完了したら特権を削除するようにします。この方法は、「特権の囲い込み」と呼ばれます。詳細は、『Oracle Solaris 10 セキュリティー開発者ガイド』を参照してください。

- バイナリラベルの変換 – ラベルをその内部表現と文字列との間で変換できます。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は、呼び出し元プロセスに `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よって所有されていない場合は、呼び出し元プロセスの実効セットに `file_owner` 特権が含まれている必要があります。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、ファイルを無関係ラベルに付け替えることもできます。

  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の方法で動作するので、ほとんどのアプリケーションはアクセス制御を省略するために特権を使用しません。

- 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、必須アクセス操作は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以外の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」を使用すれば、アプリケーションは MAC または DAC を省略できます。たとえば、`file_dac_read`、`file_dac_write`、および `file_dac_search` 特権よって DAC を省略できます。`file_upgrade_sl` および `file_downgrade_sl` 特権によって MAC を省略できます。どのような方法でアクセスを得るにしても、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いけません。

アプリケーションで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は、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

このセクションでは、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。これらの 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 の API には、次に対応するデータ型が含まれています。

- 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すことができます。`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` データ構造体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。構造体のフィールドは `variable.lower_bound` および `variable.upper_bound` と呼ばれます。

このセクションでは、次の操作に対応する API について説明します。

- Trusted Extensions システムの検出
- プロセス機密ラベルへのアクセス
- ラベル用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
- ゾーン内のラベルへのアクセス
- リモートホストタイプの取得
- ラベルと文字列の変換
- ラベルの比較

Trusted Extensions システムの検出

`is_system_labeled()` ルーチンは、Trusted Extensions システム上で実行しているかどうかを判定するために使用します。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ています。

`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になっている場合、`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については、55 ページの「`get_peer_label()` ラベル対応関数」を参照してください。

また、これらの他のインタフェースを使用して、システムにラベルが付いているかどうかを調べることもできます。

- **X** クライアント。マルチレベル機能に依存する X クライアントを作成している場合は、`XQueryExtension()` ルーチンを使用して X サーバーに `SUN_TSOL` 拡張を問い合わせます。
- シェルスクリプト。システムにラベルが付いているかどうかを調べるシェルスクリプトを作成している場合は、`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()` 関数を示しています。

```
#
#  Returns zero (success) if system is labeled (aka Trusted Extensions).
#  1 otherwise.
#
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()` ルーチンは、プロセスの機密ラベルにアクセスするために使用します。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ています。

`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、リモートプロセスの資格情報に含まれている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については、55 ページの「`get_peer_label()` ラベル対応関数」を参照してください。

ラベル用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() ルーチンは、ラベル用のメモリーの割り当てと解放に使用します。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ています。

```
m_label_t *m_label_alloc(const m_label_type_t label_type);
```
m_label_alloc() ルーチンは、ヒープ上の m_label_t データ構造体にラベルを割り当てます。ラベルの割り当ては、getlabel() や fgetlabel() などのルーチンを呼び出す前に行う必要があります。str_to_label() などの一部のルーチンでは、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は、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、ラベル用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は、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() システムコールは、ファイルのラベルを取得および設定するために使用します。次の記述には、それらのルーチンとシステムコールのプロトタイプ宣言が含まれています。

```
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、そのファイルは新しいラベルに対応するゾーンに移されます。ファイルは、その別のゾーンのルートからの相対パスである新規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 に変更した場合、そのファイルの新しいパスは /zone/restricted/documents/designdoc.odt になります。宛先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、元のファイルが削除されます。唯一の例外は、ソースおよび宛先のファイルシステムが、同一のベース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ている場合に起こります。この場合、ファイルの名前が変更されます。

プロセスでオブジェクトが作成される場合、その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上で設定します。

ファイルマネージャーアプリケーションと `setlabel` コマンドは、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を許可します。`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* で指定されるファイルのラベルを取得します。このラベルは、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、開いているファイルのラベルを取得します。

`m_label_t` 構造体を割り当てた場合は、`m_label_free()` ルーチンを使用して、割り当てたメモリーを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()` ルーチンは、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ています。

`m_range_t *getuserrange(const char *username);`
`getuserrange()` ルーチンは、指定されたユーザーのラベル範囲を取得します。範囲の下限は、ユーザーがマルチレベルデスクトップにログインしたときの初期ワークスペースラベルとして使用されます。上限あるいは認可上限は、ユーザーがラベル付けされたワークスペースに割り当てることができる使用可能なラベルの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` ファイルに指定されます。この値は `user_attr` ファイルによってオーバーライドできます。

`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。デバイスのラベル範囲を指定しない場合のデフォルトの範囲では、上限が `ADMIN_HIGH`、下限が `ADMIN_LOW` になります。

`list_devices` コマンドを使用して、デバイスのラベル範囲を表示することができます。

`list_devices(1)` のマニュアルページを参照してください。

ゾーン内のラベルへのアクセス

これらの関数は、ゾーン内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ています。

`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 `/./` および `/../` への参照を解決し、余分なスラッシュ (/) 文字を削除して、*resolved_path* で指定されたバッファーにゾーンのパス名を格納します。*bufsize* 変数は、このバッファーのサイズをバイト単位で指定します。結果となるパスには、シンボリックリンクコンポーネントも、`/./` や `/../` も一切含まれません。この関数は、大域ゾーンからのみ呼び出すことができます。

ゾーンパス名は、機密ラベル *sl* からの相対位置で示されます。存在しないゾーン名に対して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、*label* というラベルを持つゾーンのゾーン ID を返します。このルーチンでは、指定されたゾーンの状態が少なくとも `ZONE_IS_READY` である必要があります。呼び出し元プロセスのゾーンが指定され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内に存在する必要があります。

`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。このルーチンでは、指定されたゾーンの状態が少なくとも `ZONE_IS_READY` である必要があります。呼び出し元プロセスのゾーンが指定され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内に存在する必要があります。

`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、*zonename* という名前のゾーンの MAC ラベルを返します。このルーチンでは、指定されたゾーンの状態が少なくとも `ZONE_IS_READY` である必要があります。呼び出し元プロセスのゾーンが指定され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内に存在する必要があります。

`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。このルーチンでは、指定されたゾーンの状態が少なくとも `ZONE_IS_READY` である必要があります。呼び出し元プロセスのゾーンが指定され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内に存在する必要があります。返されるパス名は、呼び出し側のゾーンのルートパスからの相対位置で示されます。

`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、*label* というラベルを持つゾーンのルートパス名を返します。このルーチンでは、指定されたゾーンの状態が少なくとも `ZONE_IS_READY` である必要があります。呼び出し元プロセスのゾーンが指定され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内に存在する必要があります。返されるパス名は、呼び出し側のゾーンのルートパスからの相対位置で示されます。

`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。このルーチンでは、指定されたゾーンの状態が少なくとも `ZONE_IS_READY` である必要があります。呼び出し元プロセスのゾーンが指定され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内に存在する必要があります。返されるパス名は、呼び出し側のゾーンのルートパスからの相対位置で示されます。

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

リモートホストタイプの取得

このルーチンは、リモートホストタイプを特定します。次のルーチンの記述にはプロトタイプ宣言が含まれています。

`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特定します。*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

カードアドレスにできます。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になります。現時点では、これらの型は 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。

ラベルと文字列の変換

label_to_str() および str_to_label() ルーチンは、ラベルと文字列との変換に使用します。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ています。

```
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すると、格付け名を表示しない文字列にラベルを変換できます。この形式は公開オブジェクトへの格納に適しています。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 特権が含まれている必要があります。

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。呼び出し側では、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は、m_label_free() ルーチンを使用して、割り当てたメモリーを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は、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は通常、ウィンドウなどで表示用に使用されます。格付けされた文字列は

格納に適さない場合があります。いくつかの変換タイプが出力のために提供されています。すべての出力用タイプは、文字列が示すラベルで格付けされてい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ラベルタイプに有効であるとは限りません。

- M_LABEL は、ラベルのタイプ (機密または認可上限) に基づいたラベルの文字列です。このラベル文字列はそのラベルのレベルで格付けされるため、公開オブジェクトに格納する場合は安全ではありません。たとえば、CONFIDENTIAL などの M_LABEL 文字列は、ラベル内の単語が格付けされることが多いので、公開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。この文字列は公開オブジェクトに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と最下部の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で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 ファイル内のラベル定義を使用します。この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。このファイルには、サイト固有のラベルの定義と制約が含まれています。このファイルは /etc/security/tsol/label_encodings に保存されています。label_encodings ファイルについては、『Solaris Trusted Extensions ラベルの管理』、『Compartmented Mode Workstation Labeling: Encodings Format 』、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() ルーチンは、ラベルの比較に使用します。blinrange() ルーチンは、ラベルが指定されたラベル範囲内にあるかどうかを調べるために使用します。これらのルーチンでは、*level* は機密ラベルまたは認可上限ラベルに含まれる 1 つの格付けと一連のコンパートメントを指します。

`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* よりも完全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、ラベル *level* が指定された範囲 *range* 内にあるかどうかを判別します。

これらのルーチンでは、比較が真の場合はゼロ以外の値を返し、比較が偽の場合は値 `0` 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、59 ページの「プリンタのラベル範囲に基づいたラベルリクエストの検証」を参照してください。

ラベル関係の詳細については、17 ページの「ラベルの関係」を参照してください。

`blmaximum()` および `blminimum()` ルーチンは、指定されたラベル範囲の上限と下限を調べ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低い方であり、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使用します。

たとえば、このルーチンを使用すると、ラベル付けされた 2 つのオブジェクトから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められます。新しいオブジェクトのラベルは、ラベル付けされた元のいずれのオブジェクトよりも優位になります。

詳細は、`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 つのレベルで制限される範囲の最大下限を表し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。

詳細は、`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、ラベル付きゾーンからや他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開始できます。

プロセスでオブジェクトが作成される場合、その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き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必要があります。

例 2–1 `runwlabel` スクリプト

`runwlabel` スクリプトは最初に、指定されたプログラムを実行するラベル付きゾーンの機密ラベルを取得する必要があります。このスクリプトは、`getzonepath` コマンドを使用して、コマンド行に指定された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` コマンドで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見つけます。`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。例:

```
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
#
# Usage:
# runwlabel "my-label" my-program
#
[ ! -x /usr/sbin/zoneadm ] && exit 0    # SUNWzoneu not installed

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

# Get the zone path associated with the "my-label" zone
# Remove the trailing "/root"
zonepath=‘getzonepath "$1" | sed -e ’s/\/root$//’‘
progname="$2"
```

例 2–1 runwlabel スクリプト (続き)

```
# Find the zone name that is associated with this zone path
for zone in `zoneadm list -pi | nawk -F: -v zonepath=${zonepath} '{
        if ($4 == zonepath) {
            print $2
        }
    }'`; do

        # Run the specified command in the matching zone
        zlogin ${zone} ${progname}
    done
exit
```

次のスクリプト runinzone は、ゾーンがブート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必要があります。

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ブートし、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、ローカルゾーンのオートマウンタが起動するまで待機してから、指定されたプログラムを指定され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

# Boot the specified zone
zoneadm -z ${zonename} boot

# Run the command in the specified zone
zlogin ${zonename} /bin/demo/waitforzone ${user} ${program} ${DISPLAY}
```

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

# Wait for the local zone automounter to come up
```

例 2–2 runinzone スクリプト (続き)

```
# by checking for the auto_home trigger being loaded

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

# Now, run the command you specified as the specified user

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

◆ ◆ ◆ 第 3 章

3 ラベルのコード例

この章には、第 2 章「ラベルと認可上限」で説明されているラベル API の使用方法を示すいくつかのコード例が記載されています。

この章で扱う内容は、次のとおりです。

プロセスラベルの取得

このコード例は、このプログラムが動作しているゾーンの機密ラベルを取得し、出力する方法を示しています。

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

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

    /* allocate an m_label_t for the process sensitivity label */
    pl = m_label_alloc(MAC_LABEL);
    /* get the process sensitivity label */
    if ((retval = getplabel(pl)) != 0) {
        perror("getplabel(pl) failed");
        exit(1);
    }

    /* Translate the process sensitivity label to text and print */
```

```
    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);

    /* free allocated memory */
    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、プログラムが動作しているゾーンから継承されます。次に、このプログラム例のテキスト出力を示します。

```
Process label = ADMIN_LOW
```

テキスト出力は、`label_encodings` ファイルでの指定によって異なります。

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。

このコード例では、`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()` ルーチンも同様に使用できますが、それはファイル記述子に作用します。

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

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

    /* allocate label and get the file label specified by path */
    docLabel = m_label_alloc(MAC_LABEL);
    retval = getlabel(path, docLabel);

    /* translate the file's label to a string and print the string */
    retval = label_to_str(docLabel, &label_string, M_LABEL, LONG_NAMES);
    printf("The file's label = %s\n", label_string);

    /* free allocated memory */
    m_label_free(docLabel);
    free(label_string);
    }
```

このプログラムを実行した場合、出力はこのように表示される可能性があります。

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

ファイルの機密ラベルの設定

ファイルの機密ラベルを変更すると、そのファイルの新しいラベルに一致する新しいゾーンにそのファイルが移されます。

このコード例では、プロセスは CONFIDENTIAL ラベルで動作しています。プロセスを実行しているユーザーには、TOP SECRET 認可上限が割り当てられています。TOP SECRET ラベルは、CONFIDENTIAL ラベルよりも優位です。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、「Upgrade File Label」RBAC 承認がユーザーに必要です。

次のプログラムは upgrade-afile と呼ばれます。

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

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

   /* Create new sensitivity label value */
   if ((retval = str_to_label(string, &fsenslabel, MAC_LABEL, L_DEFAULT, &err)) != 0) {
        perror("str_to_label(MAC_LABEL, L_DEFAULT) failed");
        exit(1);
    }

   /* Set file label to new value */
   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よって異なります。

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、それはもう表示されません。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、再格付けしたファイルが表示されます。

2 つのラベル間の関係の特定

アプリケーションがさまざまな機密ラベルでデータにアクセスする場合は、アクセス操作の発生を許可する前に、コードをチェック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チェックし、システムによってアクセスが許可されているかどうかを確認します。

次のコード例は、2 つの機密ラベルの等価、優位、および完全な優位をテストする方法を示しています。このプログラムは、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、または同等であるかをチェックします。

```
#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);
}
```

このプログラムのテキスト出力は、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よって異なります。

- 「優位」には「同等」が含まれるので、ラベルが同等である場合、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も完全に優位である場合、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

このコード例では、`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対応付けは `label_encodings` ファイルで定義されます。

```
#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` ファイルで青色がラベル `CONFIDENTIAL` に対応付けられている場合、このプログラムは次を出力します。

```
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、プリンタ出力でセキュリティー情報を出力するのに役立ついくつかの変換が定義されます。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できます。

次のコード例では、label_to_str() ルーチンはラベルをヘッダーとフッター、警告のセクション、処理チャネルなどの文字列に変換します。第 4 章「印刷とラベル API」に示すように、このルーチンは 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 というプロセスラベルの場合、テキスト出力は次のようになる可能性があります。

```
            "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、『Compartmented Mode Workstation Labeling: Encodings Format 』、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、ラベル対応にする必要があるサービスタイプの 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を例として使用することによって Trusted Extensions のラベル API について説明します。

この章で扱う内容は、次のとおりです。

ラベル付き出力の印刷

通常、プリンタは共有リソースです。マルチレベル印刷では、さまざまなセキュリティーレベルで作業しているユーザーが、セキュリティーポリシーの制約に従ってプリンタを共有できます。この印刷サービスはラベル対応でもあるため、印刷されるドキュメントには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、バナーページと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と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、印刷リクエスト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。アダプタはこのラベル情報を印刷ジョブとともに渡すので、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。そのため、ほとんどの Oracle Solaris ソフトウェアアプリケーションは、変更せずに Trusted Extensions の下で動作します。Trusted Extensions のラベルベースのアクセス制限は、Oracle 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されるプロセスはどれも、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を強制するため、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も下位のラベルで実行されているために、そのリソースにアクセスできないことがあります。ただし、そのリソースにアクセスしようとすることで特別なエラー状態が生じることはありません。代わりに、アプリケーションによって `File not found` というエラーが表示されることがあります。あるいは、アクセスが許可されていない上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、十分な特権がなければ、アプリケーションはより上位のラベルを持つリソースの存在を認識できないと定めています。したがって、アプリケーションのラベルよりも上位のラベルを持つリソースにそのアプリケーションがアクセスしようとした場合、結果となるエラー状態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。「特別なエラー状態」がないことが、セキュリティー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どうかを判別しません。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、そのリソースはそのアプリケーションで使用できません。したがって、アプリケーションは、ラベル付きのリソースにアクセスするためにラベルに関する知識を持ってい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されている必要があります。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、ラベルなし環境、および複数のラベルを含む環境で基本的に同じように動作する必要があります。単一ラベル環境の例は、特定のラベルを持つ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。「ラベルなし環境」では、ラベルは明示的に設定されませんが、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

この印刷サービスはさまざまなラベルで動作するプロセスからのリクエストを受け入れるので、印刷はラベル対応である必要があります。MAC は通常、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。印刷リクエスト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、印刷サービスではこれら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します
- ラベル付き環境で印刷プロセスが実行されている場合、印刷リクエストの発行元となるネットワーク接続の資格を取得します (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します
- プリンタのラベル範囲、すなわち、プリンタがリクエストを受け入れることができるラベルの範囲を取得します
- 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どうかを判別します

`get_peer_label()` ラベル対応関数

`lp/lib/lp/tx.c` ファイルに含まれている `get_peer_label()` 関数は、Trusted Extensions でのマルチレベル印刷のロジックを実装します。以降のセクションでは、この関数について説明し、その実装を順を追って行なっていきます。

Trusted Extensions ソフトウェアでは、印刷サービスでのラベルを処理するためのロジックの大部分は、`get_peer_label()` 関数に含まれています。この関数は、`ucred_t` データ構造体に含まれるリモートプロセスの資格を取得し、その資格からラベルを抽出します。

次に、`get_peer_label()` のコードを示します。

```
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);
}
```

印刷サービスがラベル付き環境で実行されているかどうかの確認

印刷サービスは、ラベル付き環境とラベルなし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では、リモートホストのラベルがリクエストされる時期と、そのラベルが適用されるかどうかを特定する必要があります。印刷プロセスは最初に、それ自身の環境をチェックします。プロセスがラベル対応の環境で実行されているかどうかを確認します。

アプリケーションではリモートリクエストがラベル付けされているかどうかを最初に確認しないことに注意してください。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どうかを確認します。アプリケーションがラベル付き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きのリクエストを受信できません。

印刷サービスは`is_system_labeled()` 関数を使用して、プロセスがラベル付き環境で実行されているかどうかを確認します。この関数については、`is_system_labeled(3C)` のマニュアルページを参照してください。

このコード抜粋は、アプリケーションがラベル付き環境で実行されているかどうかを確認する方法を示しています。

```
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 で構成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() 関数はリモートプロセスから ucred_t 資格抽象を取得します。次に、リモートプロセスの ucred_t 構造体とピアのラベルが NULL に設定されます。資格とピアのラベルを表す値を返す関数によってそのデータ構造体が満たされます。これらのデータ構造体については、以降のセクションで説明します。

get_peer_label() ルーチン全体のソースを調べるには、55 ページの「get_peer_label() ラベル対応関数」を参照してください。

リモートホストの資格について

Oracle Solaris OS ネットワーク API は、プロセスの資格を抽象化した概念を提供します。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。これらの資格は ucred_t データ構造体によって表現されます。この構造体にはプロセスのラベルを含めることができます。

ucred API には、リモートプロセスから ucred_t データ構造体を取得するための関数が備わっています。この API には、ucred_t データ構造体からラベルを抽出するための関数もあります。

資格とリモートホストラベルの取得

リモートプロセスのラベルの取得は、2 つの手順から成るプロシージャーです。最初に、資格を取得する必要があります。次に、その資格からラベルを取得する必要があります。

資格は、リモートプロセスの ucred_t データ構造体に含まれています。ラベルは、その資格の m_label_t データ構造体に含まれています。リモートプロセスの資格を取得したあとで、その資格からラベル情報を抽出します。

getpeerucred() 関数は、リモートプロセスから ucred_t 資格データ構造体を取得します。ucred_getlabel () 関数は、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 つの手順から成るプロシージャーがコード化されています。

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

sl = ucred_getlabel(uc);
```

get_peer_label() ルーチン全体のソースを調べるには、55 ページの「get_peer_label() ラベル対応関数」を参照してください。

これらの 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。

リモートホストのラベルのほかに、リモートホストのタイプも取得できます。リモートホストのタイプを取得するには、tsol_getrhtype() ルーチンを使用します。38 ページの「リモート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

資格とリモートホストラベルを取得したあとに、アプリケーションで label_to_str() を呼び出して、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、アプリケーションで使用できます。

Trusted Extensions 印刷サービスでは、ラベルは文字列として返されることに注意してください。get_peer_label() 関数は、m_label_t データ構造体に対して 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 パラメータである 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ています。

```
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を調べるには、55 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

55 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、呼び出し側で解放する必要がある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。

- 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、リモート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。ラベル対応のアプリケーションではよくあるように、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明確に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。
- あるリソースのラベルが、別のラベル付きリソースによる特定の操作の実行を許可することを検証するため。つまり、リクエスト中のプロセスのラベルは、このプリンタがそのリクエスト中のプロセスからリクエストを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いています。

プリンタのラベル範囲に基づいたラベルリクエストの検証

印刷アプリケーションでは、ラベルを検証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種類によっては、指定された 2 つのラベルを比較する必要があります。たとえば、あるラベルが別のラベルより完全に優位であるかどうかをアプリケーションで判定することが必要な場合があります。これらのアプリケーションでは、あるラベルを別のラベルと比較する 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、一連のさまざまなラベルから印刷リクエスト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。このアプリケーションは、リモートホストから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チェックします。

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、リモートホストのラベルとプリンタのラベル範囲を照合します。このチェックは、ここに示すように 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) {
        /* str_to_label error on printer max label */
        return (0);
    }
    if ((range = getdevicerange(printer)) == NULL) {
        m_label_free(sl);
        return (0);
    }

    /* blinrange returns true (1) if in range, false (0) if not */
    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() は str_to_label() 関数を使用してその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、それによって、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チェックで適正な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 ラベルがチェックされて、リモートホストのラベル slabel が、リクエストされたデバイス (つまり、プリンタ) の範囲内にあるかどうかが判定されます。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、選択されたプリンタに対して getdevicerange() 関数を呼び出すことで取得されます。この範囲は blrange_t データ構造体として返されます。

blrange_t データ構造体に含まれるプリンタのラベル範囲は、リクエスト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 ファイルに含まれている _validate() 関数を示しています。この関数は、印刷リクエストを処理するプリンタの検索に使用されます。このコードは、ユーザー ID を許可ユーザーのセットと照合し、リクエストに関連付けられたラベルを各プリンタに関連付けられているラベル範囲と照合します。

```
/*
 * If a single printer was named, check the request against it.
 * Do the accept/reject check late so that we give the most
 * useful information to the user.
 */
if (pps) {
    (pc = &single)->pps = pps;

    /* Does the printer allow access to the user? */
    if (!CHKU(prs, pps)) {
        ret = MDENYDEST;
        goto Return;
    }

    /* Check printer label range */
    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

Trusted Extensions で構成されたシステムでは、必須アクセス制御 (MAC) および任意アクセス制御 (DAC) が適用されます。アクセス制御は、同じホスト上およびネットワークを介して通信中のプロセス間に適用されます。この章では、Trusted Extensions で構成されたシステムで使用可能なプロセス間通信 (IPC) のメカニズムの概要を示します。この章では、アクセス制御を適用する方法についても説明します。

この章で扱う内容は、次のとおりです。

マルチレベルポート情報

Trusted Extensions で構成されたシステムでは、シングルレベルポートとマルチレベルポートがサポートされます。これらのポートは、アプリケーション間の接続を作成するために使用されます。マルチレベルポートは、そのポートで定義された機密ラベルの範囲内でデータを受信できます。シングルレベルポートは、指定された機密ラベルでのみデータを受信できます。

- シングルレベルポート – 特権のない 2 つのアプリケーション間で通信チャネルが確立されます。これらの通信エンドポイントの機密ラベルが等しくなければなりません。
- マルチレベルポート – `net_bindmlp` 特権をその実効セットに含むアプリケーションと、異なる機密ラベルで動作する特権のない任意の数のアプリケーションとの間で通信チャネルが確立されます。プロセスの実効セットで `net_bindmlp` 特権を持つアプリケーションは、受信側の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、異なるラベルで動作中の 2 つの Trusted Extensions アプリケーション間で接続を確立するサーバーサイドメカニズムです。Trusted Extensions クライアントアプリケーションが、信頼されないオペレーティングシ

ステム上で異なるラベルで動作するサービスと通信できるようにする際、SO_MAC_EXEMPT ソケットオプションを使用できる場合があります。詳細は、27 ページの「MAC 除外ソケット」を参照してください。

注意 – 接続がマルチレベルの場合、アプリケーションが 1 つの機密ラベルで接続を行なったあとに別の機密ラベルでデータを送信または受信するということがないようにしてください。このような構成を行うと、データが不正な宛先に到達します。

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ための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的な操作は必要ありません。具体的には、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エンドポイントのセキュリティー属性を変更することもできません。パケットのほかのセキュリティー情報を読み取るように、パケットのラベルを読み取ることができます。ラベル情報の読み取りには ucred_getlabel() 関数が使用されます。

アプリケーションでマルチレベルポートの使用が必要な場合、そのポートはプログラム的に作成できません。そうする代わりに、アプリケーション用のマルチレベルポートを作成するようシステム管理者に求める必要があります。

マルチレベルポートの詳細については、次を参照してください。

- 『Trusted Extensions 管理者の手順』の「ゾーンとマルチレベルポート」
- 『Trusted Extensions 管理者の手順』の「How to Create a Multilevel Port for a Zone」
- 『Trusted Extensions 管理者の手順』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」

通信エンドポイント

Trusted Extensions ソフトウェアは、次に示すソケットベースのメカニズムを使用して、通信エンドポイント間の IPC をサポートします。

- Berkeley ソケット
- トランスポートレイヤーインタフェース (TLI)
- リモート手続き呼び出し (RPC)

このセクションでは、ソケット通信メカニズムおよび関連するセキュリティーポリシーの概要を示します。セキュリティーポリシーおよび該当する特権に関する具体的な情報については、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に加えて、Trusted Extensions はマルチレベルポートもサポートします。63 ページの「マルチレベルポート情報」を参照してください。

Berkeley ソケットおよび TLI

Trusted Extensions ソフトウェアは、Berkeley ソケットおよび TLI を使用した、シングルレベルポートおよびマルチレベルポート間でのネットワーク通信をサポートします。AF_UNIX ファミリのシステムコールは、完全に解決されたパス名を使用して指定された特殊ファイルを使用することによって、同一のラベルが付いたゾーン内でプロセス間通信を確立します。AF_INET ファミリのシステムコールは、IP アドレスおよびポート番号を使用して、ネットワーク経由のプロセス間接続を確立します。

AF_UNIX ファミリ

AF_UNIX ファミリのインタフェースでは、単一の特殊ファイルへの 1 つのサーバーバインドのみを確立でき、これは UNIX ドメインソケットです。AF_UNIX ファミリはマルチレベルポートをサポートしません。

UNIX ドメインソケットのように、ドアおよび名前付きパイプは、ランデブー目的の特殊ファイルを使用します。

Trusted Extensions のすべての IPC メカニズムのデフォルトポリシーは、ラベル付けされた単一ゾーン内で機能するようにすべて制約されるというものです。このポリシーの例外を次に示します。

- 大域ゾーン管理者は、所有側のゾーンより優位なラベルを持つゾーンに対して名前付きパイプ (FIFO) を使用可能にすることができます。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実行します。

  より高いレベルのゾーンで実行するプロセスは、FIFO を読み取り専用モードで開くことが許可されます。プロセスは FIFO を使用してライトダウンすることができません。
- 大域ゾーンランデブーファイルがラベル付きゾーンにループバックマウントされた場合、ラベル付き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 ソフトウェアはドアポリシーに基づいて、labeld および nscd ドアベースサービスをサポートします。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、ラベル付けされた各ゾーンにループバックマウントされることを指定しています。

AF_INET ファミリ

AF_INET ファミリでは、プロセスは特権付きポート番号または特権なしポート番号に対してシングルラベル接続またはマルチラベル接続を確立できます。特権付きの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が求められる場合、net_bindmlp 特権も必要になります。

マルチレベルポート接続を行う場合、サーバープロセスはその実効セット内に net_bindmlp 特権が必要です。代わりにシングルレベルポート接続が実行される場合、サーバープロセスはソケットに対する必須の同位書き込みアクセスが必要

で、クライアントプロセスは必須の同位読み取りアクセスが必要です。どちらのプロセスも、ファイルへの必須アクセスおよび任意アクセスが必要です。ファイルへのアクセスが拒否された場合、アクセスが拒否されたすべてのプロセスは、その実効セット内にアクセスを取得するための適切なファイル特権が必要です。

次のコード例では、マルチレベルサーバーが、接続されたクライアントのラベルを取得する方法を示しています。標準の 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 のコンテキストで、マルチレベルポートサーバーの待機中ソケットが接続リクエストを受け入れたとき、最初の引数は通常、クライアントソケットファイル記述子です。Trusted Extensions アプリケーションは、通常のアプリケーションプログラムが行うのとまったく同じ方法で `getpeerucred()` 関数を使用します。Trusted Extensions の追加関数は `ucred_getlabel()` で、これはラベルを返します。詳細は、`ucred_get(3C)` のマニュアルページを参照してください。

```
/*
 * This example shows how a multilevel server can
 * get the label of its connected clients.
 */
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));

        /*
         * Get client attributes from the socket
         */
        if (getpeerucred(clnt_fd, &uc) == -1) {
            return;
        }

        /*
         * Extract individual fields from the ucred structure
         */

        sl = ucred_getlabel(uc);

        /*
         * Security label usage here
         * .....
         */

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

RPC メカニズム

Trusted Extensions ソフトウェアは、リモート手続き呼び出し (RPC) 用のマルチレベルポートサポートを提供しています。クライアントアプリケーションはサーバーの PORTMAPPER サービス (ポート 111) に対して、特定のサービスが使用可能かどうかを照会する問い合わせを送信できます。リクエストされたサービスがサーバー上の 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、このポートをクライアントに返します。

Trusted Extensions システムでは、管理者は PORTMAPPER ポートをマルチレベルポートとして構成することで、複数のシングルレベルアプリケーションがこのサービスを使用できるようにすることが可能です。PORTMAPPER ポートがマルチレベルポートにされた場合、PORTMAPPER サービスによって割り当てられたすべての匿名ポートもマルチレベルポートになります。匿名マルチレベルポートを管理するためのプログラム可能なインタフェースまたは管理インタフェースはほかにありません。

UDP でのマルチレベルポートの使用

前のセクションで説明した PORTMAPPER サービスは、UDP を使用して実装されます。TCP とは異なり、UDP ソケットは接続指向でないため、マルチレベルポート上のクライアントに応答する際、どの資格を使用するかについて、あいまいさが発生する可能性があります。したがって、クライアントのリクエストソケットを、サーバーの応答パケットに明示的に関連付ける必要があります。この関連付けを実行するには、SO_RECVUCRED ソケットオプションを使用します。

SO_RECVUCRED が UDP ソケットに設定された場合、カーネル UDP モジュールは ucred 構造内のラベルを付属データとしてアプリケーションに渡すことができます。ucred の level および type の値はそれぞれ SOL_SOCKET および SCM_UCRED です。

アプリケーションはこの ucred 構造を、次のいずれかの方法で処理することができます。

- この ucred 構造を受信側バッファーから送信側バッファーにコピーする
- 受信側バッファーを送信側バッファーとして再使用し、ucred 構造は受信側バッファー内に残す

次のコードの抜粋は、再使用するケースを示します。

```
/*
 * Find the SCM_UCRED in src and place a pointer to that
 * option alone in dest. Note that these two 'netbuf'
 * structures might be the same one, so the code has to
 * be careful about referring to src after changing dest.
 */
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;
}
```

次のコードの抜粋は、受信側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ています。

```
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);
           }

           /*
            * Check ucred in ancillary data
            */
           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 {
                        /*
                         * You might want to extract the label from the
                         * ucred by using ucred_getlabel(3C) here.
                         */
                    }

                }

                [...]

                if (message != NULL)
                    (void) strlcpy(&inmsg[0], message, MAX_MSGLEN);
                /*
                 * Use the received message so that it will contain
                 * the correct label
                 */
                iov.iov_len = strlen(inmsg);
                ret = sendmsg(sockfd, &recv_msg, 0);
            }
}
```

◆ ◆ ◆ 第 6 章

Trusted X Window System

この章では、Trusted Extensions X Window System API について説明します。この章には、Trusted X Window System のセキュリティーポリシーおよび Trusted Extensions インタフェースの説明に使用する簡単な Motif アプリケーションも含まれています。

この章で扱う内容は、次のとおりです。

Trusted X Window System の環境

Trusted Extensions で構成されたシステムでは、共通デスクトップ環境 (CDE) の拡張バージョンである Trusted Extensions CDE (CDE) を使用します。Trusted Extensions CDE (CDE) では Trusted Extensions X Window System を使用します。Trusted Extensions X Window System 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ています。

データ転送セッションは「多インスタンス化」されます。つまり、さまざまなラベルとユーザー ID でインスタンス化されるということです。多インスタンス化は、ある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。そのような転送は、Trusted X Window System の DAC ポリシー、および同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 Window System API を使用すると、セキュリティー関連の属性情報の取得と設定を行えます。これらの API では、フォントリストおよび幅を使用してスタイルをテキスト文字列出力に適用することで、ラベルを文字列に変換するこ

ともできます。たとえば、14 ポイント、ボールドの Helvetica フォントなどです。これらの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 インタフェースで作成された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 プロトコルリクエストが出される Xlib レベルで動作します。Xlib インタフェースは、入力パラメータ値のデータ取得に使用します。
- ラベルから文字列への変換 – 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、および希望する幅です。指定したスタイルと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、77 ページの「Trusted Extensions X Window System API」を参照してください。

Trusted X Window System のセキュリティー属性

Trusted X Window System のインタフェースは、さまざまな X Window System オブジェクトのセキュリティー関連の属性情報を管理します。Motif のみを使って GUI アプリケーションを作成することもできます。Motif アプリケーションでは、Xlib オブジェクトのセキュリティー属性情報を処理するために、Motif ウィジェットの下位層にある Xlib オブジェクト ID の取得に XToolkit ルーチンを使用します。

Trusted X Window System インタフェースによってセキュリティー属性情報が取得できる X Window System オブジェクトは、ウィンドウ、プロパティー、X Window Server、およびクライアントと X Window Server 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の ID を取得する呼び出しがあります。

ウィンドウは、ユーザーへの出力を表示し、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。プロパティー名と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、32 ビットの一意の識別子とキャラクタ型の名前文字列です。

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 と機密ラベル情報で構成されます。これらの属性の一部を取得する構造体については、77 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の取得と設定を行うインタフェースについては、77 ページの「Trusted Extensions X Window System API」を参照してください。

Trusted X Window System のセキュリティーポリシー

ウィンドウ、プロパティー、およびピクセルマップの各オブジェクトには、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*」とは、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がそのオブジェクトに対してなんらかの操作を実行する必要があります。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、クライアントはそのオブジェクトを所有しています。接続リクエストを出す場合は、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) に含まれている必要があります。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同位書き込み、ネーミングウィンドウに関しては同位読み取りです。プロパティーに関しては MAC ポリシー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。次に、これらのアクションの MAC ポリシーを示します。

- 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。
- 接続リクエスト – クライアントの機密ラベルよりも 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方が優位であるか、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、クライアントで共有される情報が含まれたプロパティーを所有できます。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の機密ラベルによって区別されます。クライアントは、MAC および DAC の制限に従って、プロパティーを作成したり、ウィンドウのプロパティーにデータを格納したり、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てい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` サーバーでサポートされています。

- SPARC: `Xsun` では、このファイルは `/usr/openwin/server/etc` 内にあります。
- x86: `Xorg` では、このファイルは `/usr/X11/lib/X11/xserver` 内にあります。

これらのセクションでは、次に対する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
- オーバーライド/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にあります。ルートウィンドウはどのクライアントにも属さない公開オブジェクトですが、保護する必要があるデータが含まれています。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した少なくとも 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なった追加の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したすべてのウィンドウは同じ機密ラベルを持ちます。

オーバーライド/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オーバーライド/リダイレクトウィンドウは、もう一方のクライアントから離れて入力フォーカスを受け取ることはできません。これにより、入力フォーカスが誤った機密ラベルでのファイルへの入力を受け入れるのを防げます。オーバーライド/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がこのウィンドウを使用して別の機密ラベルでデータにアクセスすることはできません。

キーボード、ポインタ、サーバー制御

クライアントでは、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、クライアントはそのフォーカスを所有しているか、その実効セットに win_devices 特権が含まれている必要があります。

ポインタをワープするには、クライアントにポインタ制御、および移動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伴うイベントの X 座標と Y 座標の情報を取得できます。

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で情報が転送される、カット&ペーストやドラッグ&ドロップなどのユーザーレベルのウィンドウ間データ移動を制御します。転送が試みられると、選択マネージャーはその転送を取り込み、制御しているユーザーの承認を検証して、そのユーザーに確認とラベル付け情報をリクエストします。ユーザーがデータ移動を試みるときは必ず、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、その場合は選択マネージャーが表示されません。転送が MAC および DAC ポリシーを満たす場合、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も、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

クライアントで作成されないリソースは、`ADMIN_LOW` で保護されているデフォルトのリソースです。デフォルトのリソースを変更できるのは、`ADMIN_LOW` で動作しているか、適切な特権を持つクライアントに限られます。

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に読み取りアクセスと作成アクセスがありますが、書き込みアクセスまたは変更アクセスがあるのは特権クライアントのみです。76 ページの「特権操作と Trusted X Window System」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、プロトコルリクエストでデフォルトカーソルを自由に参照できます。
- 定義済みアトム – `TrustedExtensionsPolicy` ファイルには定義済みアトムの読み取り専用リストが含まれています。

ウィンドウ間のデータ移動

クライアントが選択マネージャーを介さずにウィンドウ間でデータを移動するには、その実効セットに `win_selection` 特権が含まれている必要があります。75 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 Trusted X Window System

ユーザーの介入なしでウィンドウ、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には、MAC と DAC が必要です。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には任意アクセスが必要であり、特別なタスクを実行する場合はさらに追加の特権が必要となることもあります。

オブジェクトへのアクセスが拒否される場合は、クライアントの実効セットに `win_dac_read`、`win_dac_write`、`win_mac_read`、`win_mac_write` 特権が 1 つ以上含まれていることが必要な場合があります。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。

このリストは、次のタスクの実行に必要な特権を示しています。

- ウィンドウリソースの構成と破棄 – クライアントプロセスで、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、その実効セットに `win_config` 特権が含まれている必要があります。スクリーンセーバーのタイムアウト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 – クライアントプロセスでキーボードおよびポインタ制御の取得と設定、またはポインタボタンのマッピングおよびキーマッピングの変更を行うには、その実効セットに `win_devices` 特権が含まれている必要があります。
- ダイレクトグラフィックスアクセスの使用 – クライアントプロセスで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、その実効セットに `win_dga` 特権が含まれている必要があります。
- ウィンドウラベルのダウングレード – クライアントプロセスで、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、その実効セットに `win_downgrade_sl` 特権が含まれている必要があります。
- ウィンドウラベルのアップグレード – クライアントプロセスで、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、その実効セットに `win_upgrade_sl` 特権が含まれている必要があります。
- ウィンドウのフォントパスの設定 – クライアントプロセスでフォントパスを変更するには、その実効セットに `win_fontpath` 特権が含まれている必要があります。

Trusted Extensions X Window System 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を使用してコンパイルします。

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を使用してコンパイルします。

次のセクションでは、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用のデータ型と宣言について説明します。

- 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
- ウィンドウのユーザー 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のユーザー ID へのアクセスと設定
- ワークステーションの所有者 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 Window System 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タイプを示します。値は `IsWindow`、`IsPixmap`、または `IsColormap` にできます。

  `ResourceType` は認可上限を表す型定義です。インタフェースは `m_label_t` 型の構造体をパラメータとして受け入れ、同じ型の構造体で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、これらのリソース属性が含まれています。

  ```
  typedef struct _XTsolResAttributes {
      CARD32     ouid;    /* owner uid */
  ```

```
    CARD32     uid;     /* uid of the window */
    m_label_t  *sl;     /* sensitivity label */
} XTsolResAttributes;
```

- **X11** のプロパティー属性 – `XTsolPropAttributes` 構造体には、これらのプロパティー属性が含まれています。

```
typedef struct _XTsolPropAttributes {
    CARD32     uid;     /* uid of the property */
    m_label_t  *sl;     /* sensitivity label */
} XTsolPropAttributes;
```

- **X11** のクライアント属性 – `XTsolClientAttributes` 構造体には、これらのクライアント属性が含まれています。

```
typedef struct _XTsolClientAttributes {
    int      trustflag; /* true if client masked as trusted */
    uid_t    uid;       /* owner uid who started the client */
    gid_t    gid;       /* group id */
    pid_t    pid;       /* process id */
    u_long   sessionid; /* session id */
    au_id_t  auditid;   /* audit id */
    u_long   iaddr;     /* internet addr of host where client is running */
} 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へのアクセスには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* 内の、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* 内の、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* 内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の取得と設定には、`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。

ウィンドウのユーザー ID へのアクセスと設定

ウィンドウのユーザー ID の取得と設定には、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の取得と設定には、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のユーザー 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 の取得と設定には、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の所有者 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 の取得と設定には、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のセッションの高位認可上限とセッションの最下位ラベルの設定には、XTSOLsetSessionHI() および XTSOLsetSessionLO() ルーチンが使用されま

す。セッションの高位認可上限は、ラベルビルダー GUI から選択でき、ユーザーの範囲内にある必要があります。セッションの最下位ラベルは、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すべきです。

`Status XTSOLsetSessionHI(Display *display, m_label_t *sl);`
セッションの高位認可上限は、ログイン時に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よりも優位であってはいけません。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も上位の機密ラベルで動作しているクライアントからの接続リクエストは、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、ログイン時に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、管理者によって設定されたその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優位である必要があります。この設定が変更されると、ウィンドウサーバーの機密ラベルよりも下位の機密ラベルで動作しているクライアントからの接続リクエストは、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

指定されたウィンドウをトラステッドパスウィンドウにしたり、指定され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テストしたりする場合は、`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、指定され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、指定され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テストします。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の取得と設定には、`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すべきです。

`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()` ルーチンを使用すると、クライアントは、そのクライアントとは異なる機密ラベルで、プロパティーからプロパティー情報を取得できます。最初の呼び出しで、目的の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。最後に、`enabled` プロパティーを False に設定して `XTSOLsetPolyInstInfo()` ルーチンをふたたび呼び出します。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。

Trusted X Window System インタフェースの使用

以降のセクションでは、Trusted Extensions インタフェース呼び出しを使用するコード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。この抜粋では、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ている属性である、ウィンドウのセキュリティー属性の処理に焦点を当てています。多くの場合、クライアントは適切な特権を使用し

て、別のアプリケーションで作成されたオブジェクトのセキュリティー属性を取得します。次に、クライアントはそれらの属性をチェックして、その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どうかを判別します。このセキュリティーポリシーには、DAC ポリシーと MAC の同位書き込みおよび下位読み取り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適宜エラーを生成するか、特権を使用します。特権が必要な場合については、76 ページの「特権操作と Trusted X Window System」を参照してください。

Trusted Extensions API に渡すオブジェクト ID を取得する前に、そのオブジェクトを作成する必要があります。

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の属性を返します。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
- 返される属性を受け取る `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。

このドキュメントに記載されているプログラム例は、API の紹介を中心としており、エラーチェックを行なっていないことに注意してください。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行うようにしてください。

```
/* Retrieve underlying window and display IDs with Xlib calls */
   window = XtWindow(topLevel);
   display = XtDisplay(topLevel);

/* Retrieve window security attributes */
   retval = XTSOLgetResAttributes(display, window, IsWindow, &winattrs);

/* Translate labels to strings */
   retval = label_to_str(&winattrs.sl, &plabel, M_LABEL, LONG_NAMES);

/* Print security attribute information */
   printf("Workstation Owner ID = %d\nUser ID = %d\nLabel = %s\n",
   winattrs.ouid, winattrs.uid, string1);
```

`printf` 文によって次が出力されます。

```
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、プロセスの機密ラベルを取得し、フォントリストと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ています。そのラベルを表す文字列を使用してラベルウィジェットが作成されます。プロセスの機密ラベルはウィンドウの機密ラベルと同等です。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、その文字列はクリッピングされ、クリッピングを示すインジケータが使用されます。X Window System のラベル変換インタフェースでは指定されたピクセル数にクリッピングされ、ラベルクリッピングインタフェースでは文字数にクリッピングされることに注意してください。

注 – 英語以外の言語で `label_encodings` ファイルを使用する場合、ISO 標準のコード番号が 128 以上のアクセント文字では変換が機能し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);

/* Create the font list and translate the label using it */
    italic = XLoadQueryFont(XtDisplay(topLevel),
        “-adobe-times-medium-i-*-*-14-*-*-*-*-*-iso8859-1”);
    fontlist = XmFontListCreate(italic, “italic”);
    xmstr = Xbsltos(XtDisplay(topLevel), &senslabel, width, fontlist,
        LONG_WORDS);
/* Create a label widget using the font list and label text*/
    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ています。プロセスの機密ラベルはウィンドウの機密ラベルと同等です。したがって、特権は必要ありません。

```
/* Retrieve window label */
    retval = XTSOLgetResLabel(display, window, IsWindow, &senslabel);

/* Translate labels to string and print */
    retval = label_to_str(label, &string, M_LABEL, LONG_NAMES);
    printf(“Label = %s\n”, string);
```

たとえば、`printf` 文によって次が出力されます。

```
Label = PUBLIC
```

ウィンドウラベルの設定

この例は、ウィンドウの機密ラベルを設定する方法を示しています。新しい機密ラベルは、ウィンドウの機密ラベルおよびプロセスの機密ラベルよりも優位です。クライアントでそれよりも優位なラベルを変換する場合は、その実効セットに `sys_trans_label` 特権が含まれている必要があります。また、クライアントでウィンドウの機密ラベルを変更するには、`win_upgrade_sl` 特権が必要です。

特権の使用の詳細は、『Oracle Solaris 10 セキュリティー開発者ガイド』を参照してください。

```
/* Translate text string to sensitivity label */
   retval = label_to_str(string4, &label, M_LABEL, L_NO_CORRECTION, &error);

/* Set sensitivity label with new value */
   retval = XTSOLsetResLabel(display, window, IsWindow, label);
```

ウィンドウのユーザー ID の取得

この例は、ウィンドウのユーザー ID を取得する方法を示しています。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。

```
/* Get the user ID of the window */
    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の所有者 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ています。プロセスの機密ラベルはウィンドウの機密ラベルと同等です。したがって、特権は必要ありません。

```
/* Get the user ID of the window */
    retval = XTSOLgetWorkstationOwner(display, &uid);
```

◆ ◆ ◆ 第 7 章

7 ラベルビルダー API

Trusted Extensions には、一連の Motif ベースの API が用意されています。これらのインタフェースを使用すると、ユーザー入力から有効な機密ラベルまたは認可上限を生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。ほとんどの場合、これらの API は管理アプリケーション内から呼び出されます。

ラベルビルダー GUI は、Trusted Extensions で構成されたシステムで使用されます。『Trusted Extensions User's Guide』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視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。

この章で扱う内容は、次のとおりです。

ラベルビルダー GUI 用の API

このセクションで説明する 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>
```

ラベルビルダーの例は、`-lDtTsol` および `-ltsol` ライブラリオプションを使用してコンパイルします。

ラベル GUI の作成には、次の API を使用できます。データ型とパラメータのリストは、88 ページの「対話型ユーザーインタフェースの作成」に記載されています。

```
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が含まれる `ModLabeldata` 型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` および `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ため、`tsol_lbuild_create()` には有効ではありません。

`tsol_lbuild_get()` および `tsol_lbuild_set()` ルーチンを使用すると、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。

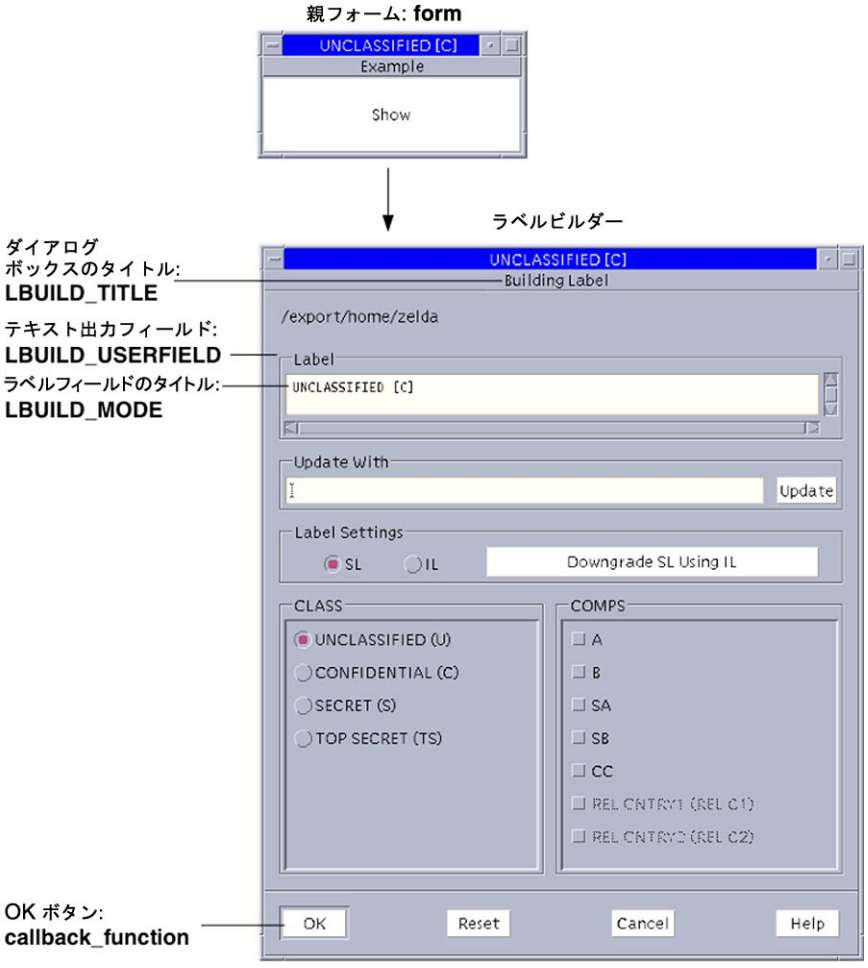
```
void *tsol_lbuild_get(ModLabelData *lbdata, lbuild_attributes
extended_operation);
```
`tsol_lbuild_get()` ルーチンは、`tsol_lbuild_create()` によって作成され `ModLabelData` 構造体に格納されるユーザーインタフェース情報にアクセスします。

```
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 `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` および `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ため、`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ものと同様の GUI を示しています。`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。ボタンのコールバックが、`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「表示」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。コールアウトは、`tsol_lbuild_create()` ルーチンに渡された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表示されるかを示しています。`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;

/* Callback passed to tsol_lbuild_create() */
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;

/* Your application-specific implementation goes here */
    printf("OK button called\n");

/* Query settings */
    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);

}

/* Callback to display dialog box upon button press */
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);

}
```

この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
```

以降のセクションでは、これらのトピックについて説明します。

- 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
- ラベルビルダーの拡張操作
- `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」、および「Update」ボタンの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チェックします。

- 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「ラベル」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用ラベルフィールドにその値を格納します。96 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を使用すると、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、これらのオプションでは、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を作成することもできます。モードによっては、これらの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、前のセクションで説明したキーボード入力や「更新」ボタンによる方法とは関係ありません。

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 ファイルで指定します。label_encodings ファイルで指定される組み合わせと制約は、無効な組み合わせをグレー表示することによって強制されます。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用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、ワークスペースの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` 特権が含まれている場合は、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されます。ユーザー作成のラベルがユーザーの範囲外にある場合も同様です。グレー表示されたボタンでは、ユーザーはタスクを完了できません。これらの制限をオーバーライド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、任意のウィジェット、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。その名前は操作を表します。このルーチンは、`ModLabelData` 型の変数を返します。

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アクティブ化します。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 (94 ページの「ラベルビルダーの拡張操作」を参照)、値 (右) 側はその値を指定します。場合によっては、値が列挙型定数になることがあります。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、指定するどの操作にも有効な値が必要です。

戻り値は、作成したばかりのダイアログボックスに関する情報が含まれるデータ構造体です。この情報は、tsol_lbuild_create() 入力パラメータと実行中のユーザーアクティビティーから得られます。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。

これらの名前と値のペアの情報にプログラム上で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。データ構造体については、96 ページの「ModLabelData 構造体」で説明しています。

次は、tsol_lbuild_create() ルーチンの呼び出しの例を示しています。

```
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);
```

ラベルビルダーの拡張操作

このセクションでは、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。パラメータで返される値には、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 構造体については、96 ページの「ModLabelData 構造体」で説明しています。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() に渡すことができます。しかし、LBUILD_WORK_SL および LBUILD_WORK_CLR 操作は、tsol_lbuild_set() または tsol_lbuild_create() に渡すことができません。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。これらの例外は、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生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「ラベル」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「ラベル」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 – ユーザーが作成している機密ラベル。ユーザーが「Update」ボタンを選択したり、対話形式でオプションを選択したりすると、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、これは tsol_lbuild_set() または tsol_lbuild_create() に拡張操作が指定されていない場合は有効ではありません。
- LBUILD_WORK_CLR – ユーザーが作成している認可上限。ユーザーが「Update」ボタンを選択したり、対話形式でオプションを選択したりすると、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、これは tsol_lbuild_set() または tsol_lbuild_create() に拡張操作が指定されていない場合は有効ではありません。
- LBUILD_X – 画面の左上隅を基準とした、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 – 画面の左上隅を基準にした、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が使用できる最上位の格付け、および関連する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形式で作成するために使用します。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の機密ラベルになります。または、実行可能ファイルに sys_trans_label 特権がある場合、この値はユーザーの認可上限になります。
- LBUILD_LOWER_BOUND – ラジオボタンとしてユーザーが使用できる最下位の格付け、および関連する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形式で作成するために使用します。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にあるかどうかをチェックします。値 1 は「チェックする」、値 0 は「チェックしない」です。ラベルが範囲外にある場合は、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。

- LBUILD_VIEW – 内部ラベル表現または外部ラベル表現のどちらを使用するかを決定します。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と最下位ラベルの内部名である ADMIN_HIGH と 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。次の表では、ModLabelData の各フィールドについて説明します。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、関連した拡張操作と有効な値を tsol_lbuild_set() または tsol_lbuild_get() の呼び出しに指定することによってアクセスできます。拡張操作については、94 ページの「ラベルビルダーの拡張操作」を参照してください。

表 7–1 ModLabelData 構造体

拡張操作または説明	データ型	フィールド	コメント
LBUILD_CHECK_AR	int	check_ar	
LBUILD_MODE	int	mode	
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	

表 7–1 ModLabelData 構造体 (続き)

拡張操作または説明	データ型	フィールド	コメント
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

ユーザーインタフェースで使用される「Help」ボタンやその他のウィジェットには、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コンテンツを隔離してインターネットからの攻撃を防ぐように構成されています。

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。むしろ、このプロトタイプは、マルチレベルポートをどのように使用すれば、ラベルの境界を越えてURLリクエストをプロキシできるかを示すよう意図されています。より完全なソリューションにするには、認証、データフィルタリング、監査などを組み込みます。

このプロトタイプの主な実装は管理手法によるものです。このプロトタイプは、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。管理手法による例のほかに、プログラムによるいくつかの方法を使用しても安全なWebブラウジングプロトタイプを設定できます。

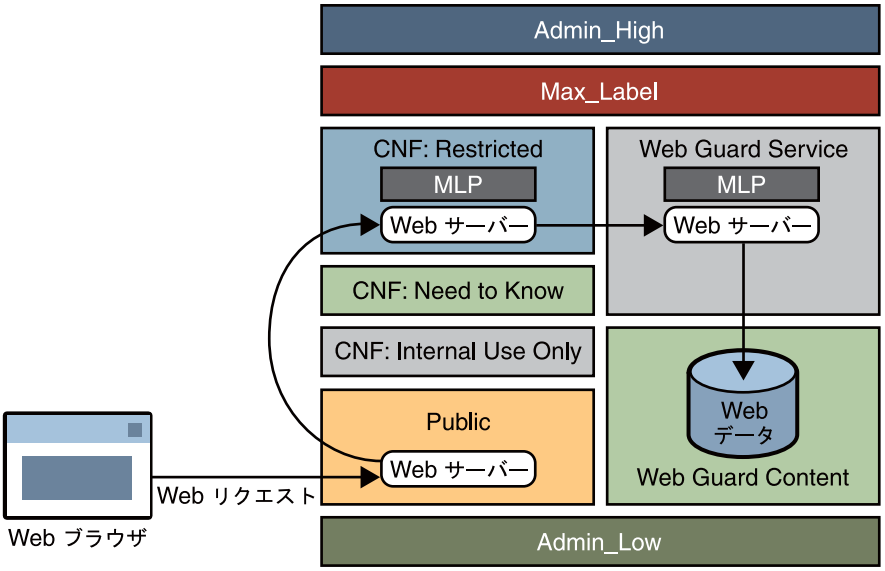
この章で扱う内容は、次のとおりです。

管理手法による**Web**ガードプロトタイプ

このセクションでは、Webサーバーとその Webコンテンツを隔離してインターネットからの攻撃を防ぐ安全な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活用して、保護されたWebサーバーとWebコンテンツへのアクセスを制限する2段階のフィルタを構成します。このプロトタイプは、管理手法によってのみ実装されました。プログラミングは必要ありませんでした。

次の図は、マルチレベル環境での Web ガードプロトタイプの構成を示しています。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優位性を表し、横の関係は無関係ラベルを表します。

図 8–1 Web ガードの構成



Web リクエストは、`public` ゾーン内に構成されている Web サーバーに入り、`restricted` ゾーン内に構成されている Web サーバーに渡されます。

`restricted` ゾーンでは、マルチレベルポート (MLP) を使用して `public` ゾーンのポート `8080` でリクエストを待機します。この Web サーバーはリクエストを `webservice` ラベル付きゾーンに渡します。

`webservice` ゾーンでも MLP を使用して `restricted` ゾーンのポート `80` でリクエストを待機し、`webcontent` ラベル付き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り、その Web コンテンツは、`/export/home` ファイルシステムに格納されています。そのファイルシステムはほかのすべてのラベル付き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にあるときは、そのゾーンで実行されているプロセスはありません。したがって、そのゾーンは基本的に `webservice` ゾーンに直接接続されたディスクドライブになります。

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

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

   2 つの新しいラベル (WEB GUARD SERVICE および WEB GUARD CONTENT) を構成するようにデフォルトの label_encodings ファイルを更新します。101 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

   プライベート IP アドレスおよび MLP を restricted および webservice ラベル付きゾーンに構成します。104 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 のすべてのゾーンに 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。105 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

2 つの新しいラベル (WEB GUARD SERVICE および WEB GUARD CONTENT) を構成するようにデフォルトの label_encodings ファイルを更新します。デフォルトファイルの一部である SANDBOX ラベルを WEB GUARD CONTENT ラベルとして機能す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 インストールの上に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、新しい label_encodings ファイルをアクティブ化します。

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

次に、Web ガードプロトタイプで使用される label_encodings ファイルを示します。

```
* ident    "@(#)label_encodings.simple    5.15    05/08/09 SMI"
*
* Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
* Use is subject to license terms.
*
* This example shows how to specify labels that meet an actual
* site's legal information  protection requirements for
* labeling email and printer output. These labels may also
* be used to enforce mandatory access control checks based on user
* clearance labels and sensitivity labels on files and directories.
```

```
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;

* Local site definitions and locally configurable options.

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;

* End of local site definitions
```

label_encodings ファイルの詳細は、『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 アドレスも割り当てます。各プライベート IP アドレスに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を関連付けます。

次の表は、それぞれのラベル付きゾーンのネットワーク構成を示しています。

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する必要があります。public ゾーンなどの既存のゾーンをクローニングできます。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 という IP アドレスと bge0 というインタフェースを 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きゾーンに IP アドレスとネットワークインタフェースを指定したら、Solaris Management Console を使用して表の残りの値を構成します。このツールを使用するときは、必ず Scope=Files および Policy=TSOL でツールボックスを選択してください。

これらの手順に従ってゾーンの構成を終了します。

1. スーパーユーザーとして Solaris Management Console を起動します。

   ```
   # smc &
   ```
2. ナビゲーションパネルから「このコンピュータ」を選択して、「システム構成」アイコンをクリックします。
3. 「コンピュータとネットワーク」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」メニューから「コンピュータを追加」を選択します。
5. proxy ホストおよび webeservice ホストのホスト名と IP アドレスを追加します。
6. ナビゲーションパネルから、「トラステッドネットワークゾーン」を選択します。

   場合によっては、列を拡張する必要があります。ゾーン名がリストに表示されない場合は、「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. 各ゾーンにそれぞれのラベルを割り当て、「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。

   表に記載された情報に基づいて 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 Management Console を終了します。

Solaris Management Console 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。ここで、*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 ファイルを更新します。

- public ゾーン – 次のように、ServerName キーワードに対してサーバーの 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。そして、次のように、ServerName キーワードに対してこのゾーンの 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 コンテンツへのリクエストの種類を制限するために、禁止用語フィルタなど、Web リクエストのフィルタリングを設定することが必要な場合もあります。

- webservice ゾーン – 次のように、ServerName キーワードに対してこのゾーンの 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> 要素で 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">
  ```

各ラベル付きゾーンで 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 ディレクトリ内に demo ディレクトリを作成してから、その demo ディレクトリ内にテストで使用する index.html ファイルを作成します。

webservice ゾーンのブート時に、/export/home ディレクトリが lofs を使用してそのゾーンに自動的にマウントされます。webcontent ゾーンは動作可能状態にしておくだけで十分です。

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

ゾーンが動作可能状態にあるときは、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。このような方法で 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 ゾーン内のブラウザから、または PUBLIC ラベルで動作しているリモートブラウザから、次の URL を入力します。

`http://server-name/demo`

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

Web ガードフローを省略できないことに注意してください。`webservice` ゾーン内の Web サーバーは、`public` ゾーンからも、どのリモートホストからもパケットを受信できません。`webcontent` ゾーンが動作可能状態にあるので、Web コンテンツを変更することはできません。

下位レベルの信頼できないサーバーへのアクセス

クライアントでは、ラベルなしシステム上のサーバーにアクセスできることが必要な場合があります。「ラベルなしシステム」とは、Trusted Extensions ソフトウェアを実行しないシステムです。そのような場合は、マルチレベルポートを使用できません。それらは大域ゾーンまたはラベル付きゾーンで動作する特権サーバーに制限されるからです。

たとえば、ブラウザが `INTERNAL` ゾーンで動作しているとします。`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動作している Web サーバーにアクセスしてみます。デフォルトでは、そのようなアクセスは許可されません。ただし、HTTP リクエストを `PUBLIC` の Web サーバーに転送する特権プロキシサーバーを記述できます。このプロキシは、`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リクエストを送信したり、そのサービスからの応答をリクエスト元に返したりすることが可能になります。

注 – `SO_MAC_EXEMPT` オプションの使用は保護されていないダウングレードチャネルを意味するので、十分に注意するようにしてください。`SO_MAC_EXEMPT` オプションは、呼び出し元プロセスの実効セットに `PRIV_NET_MAC_AWARE` 特権が含まれていない限り設定できません。そのようなプロセスでは、上位レベルのデータが下位レベルのサービスに漏れないように、それ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るのを禁止するために、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` の使用法を示しています。`SO_MAC_EXEMPT` オプションの設定方法を示すために、`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;

  /* If PRINT is non-NULL, print the "Connecting to..." line, with
     PRINT being the host name we're connecting to.  */
  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);
      }

  /* Store the sockaddr info to SA.  */
  sockaddr_set_data (sa, ip, port);

  /* Create the socket of the family appropriate for the address.  */
  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) {
    /* In case of error, we will go on anyway... */
    int err = setsockopt (sock, IPPROTO_IPV6, IPV6_V6ONLY, &on, sizeof (on));
  }
#endif
```

◆ ◆ ◆ 第 9 章

9 Solaris Trusted Extensions ラベル API のための試験的な Java バインディング

この章では、Trusted Extensions ソフトウェアで提供されているラベルアプリケーションプログラミングインタフェース (API) を反映した、Java のクラスおよびメソッドの試験的なセットについて説明します。Trusted Extensions ラベル API の Java 実装を使用する目的は、ラベルに対応したアプリケーションを作成することです。そのため、Trusted Extensions で提供されているすべてのラベル API が Java 実装の一部になるというわけではありません。

これらの試験的な Java API (Java バインディング) を示すことによって、Trusted Extensions の機能を Java 開発環境に拡張する方法を説明します。

注意 – これらの試験的な Java バインディングは、Trusted Extensions ソフトウェアでサポートされる一部分ではありません。

この章で扱う内容は、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する、ラベルに対応したアプリケーションを作成するための未利用リソースです。これらの試験的な Java バインディングによって、システム監査ログ生成やシステムリソース制御といった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ことによって、Java アプリケーションが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。

Trusted Extensions は、ラベルデーモン labeld を介してラベルサービスを提供します。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するプロセスから利用できます。

この章で説明する Java バインディングは、一部の Trusted Extensions ラベル API の Java Native Interface (JNI) 実装です。試験的な JNI コードは Trusted Extensions ラベルライブラリ関数を呼び出し、一部のラベル機能を 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 で記述されたプライベート JNI インタフェースを呼び出し、これが今度は Trusted Extensions API を呼び出します。たとえば、SolarisLabel.dominates メソッドは、bldominates() ルーチンを呼び出す C で記述されたプライベート JNI インタフェースを呼び出します。これらの試験的な 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 の詳細については、Java Native Interface のドキュメント (http://download.oracle.com/javase/1.5.0/docs/guide/jni/)を参照してください。

試験的 Java ラベルインタフェースの構造

Trusted Extensions ラベル API の JNI 実装では、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 抽象クラス
  - ClearanceLabel サブクラス
  - SensitivityLabel サブクラス
- Range クラス

SolarisLabel 抽象クラス

SolarisLabel 抽象クラスは、Trusted Extensions ラベルに関連する共通メソッドおよびネイティブメソッドのための基礎を提供します。SensitivityLabel および ClearanceLabel サブクラスは、この抽象クラスからメンバーを継承します。機密ラベルと認可上限ラベルを作成するための静的ファクトリも、抽象クラスによって提供されます。

エラーが発生したとき、静的ファクトリおよびメソッドは例外を送出し、必須アクセス制御に関連したエラーがサイレントで発生しないようにします。

この抽象クラスは、ラベルを比較し、ラベルを文字列に変換するために使用される、次の汎用メソッドを定義します。

- dominates
- equals
- setFileLabel
- strictlyDominates

- toColor
- toInternal
- toRootPath
- toString
- toText
- toTextLong
- toTextShort

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

残りのメソッド (toText、 toInternal、および toColor) は、Trusted Extensions で現在使用な label_to_str() ルーチンと機能が関連しています。これらのメソッドにより、特定のタイプの文字列にラベルを変換できます。プロセスおよびオブジェクトのラベル関係によっては、人間が読むことができる形式にラベルを変換するための特権が、実効セット内に必要になる場合もあります。たとえば、Java Virtual Machine (JVM) のプロセスは、そのプロセスよりも優位なラベルを変換する場合、sys_trans_label 特権で実行する必要があります。

SolarisLabel 抽象クラスには、次の静的ファクトリも含まれています。

- getClearanceLabel
- getFileLabel
- getSensitivityLabel
- getSocketPeer

getSensitivityLabel または getClearanceLabel にラベルとして渡す文字列は、次のいずれかの形式が可能です。

- 人間が読むことのできるラベルの形式 (PUBLIC など)
- ラベルの内部形式 (0x0002-08-08 など)

ラベルの内部形式は実際のラベルを表示しないため、内部形式のみが保管およびネットワーク接続経由の送信に適しています。詳細は、39 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 抽象クラスを拡張します。これらの各サブクラスは、SolarisLabel 抽象クラスによって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 抽象クラスを拡張し、getMaximum および getMinimum メソッドを定義します。これらのメソッドは、それぞれ最小の上限と最大の下限を表す 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 抽象クラスを拡張し、getMaximum および getMinimum メソッドを定義します。これらはそれぞれ最小の上限と最大の下限を表す SensitivityLabel オブジェクトを返します。

SensitivityLabel サブクラスは、ラベル付けされたプリンタバナーページに適した情報を提供する次のメソッドを導入します。

- toCaveats
- toChannels
- toFooter
- toHeader
- toProtectAs

Range クラス

Range クラスは、Trusted Extensions のラベル範囲の Java バージョンを表します。

このクラスは、ラベル範囲から上限および下限のラベルを取得し、あるラベルが、指定されたラベル範囲に入っているかどうかを判別するために使用する、次の汎用メソッドを定義します。

- getLower
- getUpper
- inRange

Range クラスには、範囲オブジェクトを作成する次の静的ファクトリも含まれています。

- getDeviceRange
- getLabelRange
- getUserRange

静的ファクトリ getDeviceRange および getUserRange はそれぞれ、指定されたデバイスおよび指定されたユーザーについての範囲に基づいて範囲オブジェクトを作成します。getLabelRange 静的ファクトリを使用すると、範囲の上限および下限の指定が可能なラベル範囲を作成できます。

Java バインディング

Trusted Extensions ラベル API の Java 実装を使用する目的は、ラベルに対応したアプリケーションを作成することです。そのため、Trusted Extensions で提供されているすべてのラベル API が Java 実装の一部になるというわけではありません。

この章に紹介されている 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を模倣したものです。

- 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に対する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
- ゾーン内のラベルへのアクセス
- リモートホストタイプの取得
- ラベルと文字列の変換
- 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いているかどうかを判別するためのメソッドは含まれていません。Trusted Extensions が使用可能でない場合、ライブラリのロードに失敗します。

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ためのメソッドは含まれていません。Trusted Extensions では、ラベル付けされたゾーン内で実行するプロセスにはゾーンと同じラベルが付きます。

ラベルオブジェクトに対する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「不要データの収集」機能を活用します。そのため、35 ページの「ファイルのラベルの取得と設定」に記載されているラベル API について実行するような、ラベルオブジェクトによって使用されるメモリーの明示的な解放は不要です。

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、ファイルのラベルを取得および設定します。getFileLabel 静的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。ファイルのラベルを指定された別のラベルに設定するには、ファイルの File オブジェクトに対して setFileLabel メソッドを使用します。

ファイルの機密ラベルを取得する以外に、getSocketPeer 静的ファクトリによって、ソケットのピアエンドポイントの機密ラベルを取得できます。

getFileLabel 静的ファクトリおよび setFileLabel メソッドは、getlabel() システムコールおよび setflabel() ルーチンにそれぞれ対応します。詳細は、35 ページの「ファイルのラベルの取得と設定」と、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。

次の説明には、静的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ています。

`public static SensitivityLabel getFileLabel(java.io.File file)`
: getFileLabel 静的ファクトリは、*file* によって指定される 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 静的ファクトリは、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);
```

  次のサンプルコードは、ポート 9090 上にサーバーソケットを作成し、受け入れた接続のピアエンドの機密ラベルを取得する方法を示しています。このコード例は、取得されたソケットピアラベルの内部形式、人間が読むことのできる形式、色、およびルートパスも出力します。

```
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` メソッドは、指定されたファイルの機密ラベルを指定されたラベルに変更します。ファイルの機密ラベルが変更されると、そのファイルは新しいラベルに対応するゾーンに移動します。ファイルは、その別のゾーンのルートからの相対パスである新規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` に変更した場合、ファイルの新しいパスは `/zone/restricted/documents/designdoc.odt` になります。宛先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変更する方法を示しています。

```
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、元のファイルが削除されます。唯一の例外は、ソースおよび宛先のファイルシステムが、同一のベース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ている場合に起こります。この場合、ファイルの名前が変更されます。

ファイルのラベルを変更するには、Java 仮想マシンが適切な特権 (`file_upgrade_sl` または `file_downgrade_sl`) で実行している必要があります。

`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` 静的ファクトリはラベル範囲オブジェクトを作成します。`getUserRange` および `getDeviceRange` 静的ファクトリは、それぞれユーザーおよびデバイスのラベル範囲オブジェクトを取得します。`getUpper` および `getLower` メソッドは、それぞれ範囲の上側および下側のラベルを取得するために使用されます。また、`inRange` メソッドは、指定されたラベルが範囲内にあるかどうかを判別します。`inRange` メソッドの詳細については、121 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` 静的ファクトリは、`getuserrange()` および `getdevicerange()` ルーチンに対応します。詳細は、36 ページの「ラベル範囲の取得」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説明には、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` 静的ファクトリは、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。デバイスのラベル範囲が指定されていない場合、デフォルトの範囲は `ADMIN_HIGH` が上限で、`ADMIN_LOW` が下限です。

  `list_devices` コマンドを使用して、デバイスのラベル範囲を表示することができます。`list_devices`(1) 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` 静的ファクトリは、ラベル範囲を作成します。静的ファクトリは、範囲の下限値と、上限値または認可上限をパラメータとして指定します。*upper* が *lower* より優位でない場合、例外が送出されます。

`public L getLower()`
: `getLower` メソッドは、範囲の下側部分を返します。

`public U getUpper()`
: `getUpper` メソッドは、範囲の上側部分を返します。

`public static Range getUserRange(java.lang.String user)`
`getUserRange` 静的ファクトリは、指定されたユーザーのラベル範囲を取得します。範囲の下限は、ユーザーがマルチレベルデスクトップにログインしたときの初期ワークスペースラベルとして使用されます。上限あるいは認可上限は、ユーザーがラベル付けされたワークスペースに割り当てることができる使用可能なラベルの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` ファイルに指定されます。この値は `user_attr` ファイルによってオーバーライドできます。

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

ゾーン内のラベルへのアクセス

次の説明には、メソッドのプロトタイプ宣言が含まれています。

`public final java.lang.String toRootPath()`
このメソッドは、指定された機密ラベルについてのゾーンのルートパス名を返します。

次のコードの抜粋は、`PUBLIC` の機密レベルについてのルートパスを取得する方法を示しています。

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

このメソッドは、無効なラベルが指定されたり、指定されたラベルに対してゾーンが構成されていない場合に、`java.io.IOException` を送出します。

このメソッドは、`getzonerootbylabel()` ルーチンを模倣したものです。`getzonerootbylabel(3TSOL)` のマニュアルページを参照してください。
37 ページの「ゾーン内のラベルへのアクセス」も参照してください。

リモートホストタイプの取得

Trusted Extensions ラベル API の Java 実装には、リモートホストタイプを取得するためのインタフェースが含まれていません。

ラベルと文字列の変換

`SolarisLabel` 抽象クラスには、ラベルと文字列を変換するためのメソッドが含まれており、このメソッドはそのサブクラスによって継承されます。

これらのメソッドは、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なるようラベルを変換できます。この形式は、公開オブジェクトでのラベルの保存に適しています。

実行中の Java 仮想マシンは、変換されるラベルよりも優位であるか、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いています。

次のメソッドは、label_to_str() ルーチンを模倣したものです。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 による使用に適しています。

public final java.lang.String toInternal()
: このメソッドは、公開オブジェクトで安全に保管できるラベルの内部表現を返します。内部変換をあとで構文解析することで、同じ値に戻すことができます。これは toString メソッドを使用する場合と同じです。

  これらの 2 つのメソッドは、エラーを処理する方法が異なります。toInternal メソッドでエラーが発生すると、java.io.IOException が返されます。ただし、toString メソッドでエラーが発生すると、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、toInternal メソッドを使用する場合と同じです。

  これらの 2 つのメソッドは、エラーを処理する方法が異なります。toString メソッドでエラーが発生すると、null が返されます。ただし、toInternal メソッドでエラーが発生すると、java.io.IOException が返されます。

public java.lang.String toText()
: このメソッドは、人間が読むことのできる SolarisLabel オブジェクトのテキスト文字列を返します。

public java.lang.String toTextLong()
: このメソッドは、人間が読むことのできる SolarisLabel オブジェクトの長いテキスト文字列を返します。

public java.lang.String toTextShort()
: このメソッドは、人間が読むことのできる SolarisLabel オブジェクトの短いテキスト文字列を返します。

次のメソッドは、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を模倣したものです。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、バナーページの注意セクションに適した、人間が読むことのできるテキスト文字列を返します。

このメソッドは `SensitivityLabel` オブジェクトについてのみ使用可能で、`ClearanceLabel` オブジェクトについては使用できません。

`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、人間が読むことのできるテキスト文字列を返します。

このメソッドは `SensitivityLabel` オブジェクトについてのみ使用可能で、`ClearanceLabel` オブジェクトについては使用できません。

`public java.lang.String toFooter()`
このメソッドは、機密ラベルとして適切に使用できる、人間が読むことのできるテキスト文字列を返します。この機密ラベルはバナーおよび後続ページの下部に表示されます。

このメソッドは `SensitivityLabel` オブジェクトについてのみ使用可能で、`ClearanceLabel` オブジェクトについては使用できません。

`public java.lang.String toHeader()`
このメソッドは、機密ラベルとして適切に使用できる、人間が読むことのできるテキスト文字列を返します。この機密ラベルはバナーおよび後続ページの上部に表示されます。

このメソッドは `SensitivityLabel` オブジェクトについてのみ使用可能で、`ClearanceLabel` オブジェクトについては使用できません。

`public java.lang.String toProtectAs()`
このメソッドは、バナーページのページダウングレードセクションに適した、人間が読むことのできるテキスト文字列を返します。

このメソッドは `SensitivityLabel` オブジェクトについてのみ使用可能で、`ClearanceLabel` オブジェクトについては使用できません。

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

次のサンプルコードは、Java バインディングを使用して、50 ページの「プリンタバナー情報の取得」に記載されているものに類似したバナーページを作成する方法を示しています。

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

/*
 * Banner page example
 */
public class PrintTest1
```

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)

```
{

   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) {
         e.printStackTrace();
      }
   }
}
```

プロセスラベル TOP SECRET A B SA の場合、テキスト出力は次のようになることがあります。

```
        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、toColor などのメソッドでは、文字列からラベルへの変換は行いません。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 静的ファクトリをそれぞれ呼び出す必要があります。次の静的ファクトリは、str_to_label() ルーチンを模倣したものです。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)
この静的ファクトリは、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)
この静的ファクトリは、指定され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 メソッドは、ラベルを比較するために使用され、blequal()、bldominate()、および blstrictdom() ルーチンに対応します。inRange メソッドは、ラベルが指定されたラベル範囲内にあるかどうかを判別するために使用され、blinrange() ルーチンに対応します。これらのメソッドで、ラベルは機密ラベルまたは認可上限ラベルの格付けおよびコンパートメントのセットを表します。詳細は、40 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 メソッドは 2 つのラベルを比較して、1 つのラベルが別のラベルよりも優位かどうかを判断します。

次のサンプルコードは、比較を行う方法を示しています。CNF : INTERNAL ラベルが PUBLIC ラベルより優位であることを確認するために比較されます。

```
SensitivityLabel sl = SolarisLabel.getSensitivityLabel("CNF : INTERNAL");
boolean isDominant = sl.dominates(SolarisLabel.getSensitivityLabel("PUBLIC"));
```

public boolean equals(java.lang.Object obj)
equals メソッドは 2 つのラベルを比較して、それらが等しいかどうかを判断します。

次のサンプルコードは、比較を行う方法を示しています。

```
SensitivityLabel sl = SolarisLabel.getSensitivityLabel("CNF : INTERNAL");
boolean isSame = sl.equals(SolarisLabel.getSensitivityLabel("PUBLIC"));
```

public boolean Range inRange(SensitivityLabel label)

inRange メソッドは、指定され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 クラスに所属します。

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にあることを確認する方法を示しています。

```
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 メソッドは 2 つのラベルを比較して、1 つのラベルが別のラベルよりも完全に優位であるかどうかを判断します。あるラベルが別のラベルよりも完全に優位であるとき、そのラベルは別のラベルよりも優位ですが、別のラベルと等しくはありません。

次のサンプルコードは、比較を行う方法を示しています。CNF : INTERNAL ラベルが PUBLIC ラベルより完全に優位であることを確認するために比較されます。

```
SensitivityLabel sl = SolarisLabel.getSensitivityLabel("CNF : INTERNAL");
boolean isStrictlyDominant =
    sl.strictlyDominates(SolarisLabel.getSensitivityLabel("PUBLIC"));
```

ラベル関係の詳細については、17 ページの「ラベルの関係」を参照してください。

getMaximum および getMinimum メソッドは、指定されたラベル範囲の最小の上限および最大の下限を判別するために、それぞれ使用されます。これらのメソッド

は、blmaximum() および blminimum() ルーチンの機能を反映したものです。詳細は、40 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、2 つの異なるラベル付きオブジェクトからの情報を組み合わせた新しいラベル付きオブジェクトを作成するとき、getMaximum メソッドを使用して、使用するラベルを判別します。新しいオブジェクトのラベルは、ラベル付けされた元のいずれのオブジェクトよりも優位になります。各メソッドは、ClearanceLabel および SensitivityLabel サブクラスによって定義されます。

`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優位となることができるもっとも低いラベルである新しい認可上限ラベルオブジェクトを作成します。結果として作成されるオブジェクトは、範囲の最小の上限です。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 つのラベルオブジェクトの方が優位となる、もっとも高いラベルである新しい認可上限ラベルオブジェクトを作成します。結果として作成されるオブジェクトは、範囲の最大の下限です。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優位となることができるもっとも低いラベルである新しい機密ラベルオブジェクトを作成します。結果として作成されるオブジェクトは、範囲の最小の上限です。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 つのラベルオブジェクトの方が優位となる、もっとも高いラベルである新しい機密ラベルオブジェクトを作成します。結果として作成されるオブジェクトは、範囲の最大の下限です。getMinimum は、機密ラベルの内部形式でオブジェクトを返します。

次の表に、getMaximum および getMinimum メソッドからのラベル入力と出力を示します。

表 9–1 getMinimum および getMaximum メソッドの使用

入力ラベル	**getMinimum** の出力	**getMaximum** の出力
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

表 9–1 getMinimum および getMaximum メソッドの使用 (続き)

入力ラベル	**getMinimum** の出力	**getMaximum** の出力
SECRET A B TOP SECRET	SECRET	TOP SECRET A B
SECRET A TOP SECRET B	SECRET	TOP SECRET A B

◆ ◆ ◆ 付 録 A

プログラマーのリファレンス

この付録では、ラベルに対応したアプリケーションの開発、テスト、および Trusted Extensions ソフトウェアを使用する環境にリリースすることに関する情報を見つけるための場所について説明します。

この付録では、次のトピックを扱います。

ヘッダーファイルの場所

大半の 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。ほか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その場所	インタフェースのカテゴリ
`/usr/dt/include/Dt/label_clipping.h`	X11 ラベル変換と、フォントリストを伴う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 Window System
`/usr/include/libtsnet.h`	信頼された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名で使用される省略形

Trusted Extensions のインタフェース名およびデータ構造名の多くは、次に示す短い省略形を使用します。これらの名前の省略形を知っておくと、インタフェースまたは構造の目的を理解するのに役立ちます。

表 A–1 Trusted Extensions API で使用される名前の省略形

省略形	名前
attr	属性
b	バイナリ形式
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	リエントラント
res	リソース
s	文字列
sec	セキュリティー
sl	機密ラベル
tp	トラステッドパス
tsol	Trusted Extensions
xtsol	Trusted X11 Server

アプリケーションの開発、テスト、およびデバッグ

アプリケーションの開発、テスト、およびデバッグは、隔離された開発システム上で行い、ソフトウェアバグおよび不完全なコードがメインシステムのセキュリティーポリシーを低下させないように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やセキュリティーチェックをバイパスするコード) を削除してください。
- アプリケーションデータの操作にセキュリティー上の問題がないか管理者がインストール前に検査できるように、操作を簡単に追跡できるようにしておきます。
- すべてのプログラミングインタフェースに対してリターンコードをテストします。呼び出しに失敗すると、予期できない結果が生じる可能性があります。予期しないエラー条件が発生した場合、アプリケーションを常に終了する必要があります。
- アプリケーションを実行することが予想されるすべての役割およびすべての機密ラベルで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。
  - プログラムが役割でなく通常の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ことが意図されているラベルで、コマンド行からプログラムを開始します。
  - プログラムが役割から実行される場合、大域ゾーンのコマンド行からプログラムを開始するか、プログラムが実行されることが意図されているラベルでユーザー役割からプログラムを開始します。
- アプリケーションが必要なすべての特権を備えているかどう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。このタイプのテストでは、アプリケーションが実行してはなら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判断できます。
- 特権の使用がセキュリティーに及ぼす影響を把握します。アプリケーションが特権を使用することによって、システムのセキュリティーを低下させることがないようにしてください。
- 特権のブラケット化の適切な実施方法に従ってください。

  『Oracle Solaris 10 セキュリティー開発者ガイド』を参照してください。
- `SUNWspro debugger` または `dbx` コマンドを使用して特権付きアプリケーションをテストする場合、実行中のプロセスにデバッガを接続する前にデバッガを開始します。コマンド名を引数として使用してデバッガを開始することはできません。

アプリケーションのリリース

テストおよびデバッグが完了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。アプリケーションは CDE 操作またはソフトウェアパッケージとして提出できます。アプリケーションで特権が使用される場合、システム管理者は、アプリケーションのソースコードと、ユーザーが指定する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、特権の使用によりシステムセキュリティーが低下しないことが確認されます。

注意 – 新しい監査イベント、監査クラス、またはアプリケーションが使用する X Window System プロパティーをシステム管理者に通知してください。システム管理者は、これらの項目を正しいファイルに配置する必要があります。詳細は、第 6 章「Trusted X Window System」を参照してください。

CDE 操作の作成

CDE 操作は、ユーザーまたは役割によってワークスペースから開始されます。この操作は、ユーザーまたは役割のプロファイルに割り当てられた特権を継承します。「*CDE* 操作」は、アプリケーションの実行やデータファイルのオープンなどのデスクトップタスクを自動化するために、アプリケーションマクロや API のように機能する命令のセットです。Trusted Extensions で構成されたシステム上で、アプリケーションはワークスペースから CDE 操作として開始されます。CDE 操作を作成する方法の説明は、『Solaris Common Desktop Environment: Advanced User's and System Administrator's Guide 』にあります。

注 – CDE 操作を作成するときは、`f.exec` でなく `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 操作を適切なプロファイルに配置し、必要なすべての特権を CDE 操作に割り当てます。ユーザーはプログラムで使用する特権の一覧を示し、アプリケーションが実行されることを意図したラベルを示し、必要となる実際のユーザー 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、特権と実際のユーザー ID およびグループ ID を、プロファイル内の CDE 操作に割り当てます。

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、『アプリケーションパッケージ開発者ガイド』を参照してください。パッケージのインストールに関する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

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) の一覧と、それらの使用法の相互参照を示します。宣言はセキュリティートピック別にグループ化されています。

この付録では、次のトピックを扱います。

プロセスセキュリティー属性フラグ API

次の Oracle Solaris API は Trusted Extensions の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 についての紹介は、第 2 章「ラベルと認可上限」にあります。サンプルコードは第 3 章「ラベルのコード例」にあります。詳細に説明した例は、第 4 章「印刷とラベル API」にあります。

次に、ラベルに関連した API のタイプの一覧と、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);`
- ゾーン内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);`
- リモートホストタイプの取得
  - `tsol_host_type_t tsol_getrhtype(char *hostname);`
- セキュリティーラベルへのアクセスおよび変更

- `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 の詳細は、第 6 章「Trusted X Window System」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 はリモート手続き呼び出し (RPC) 用のインタフェースを提供しません。RPC インタフェースは Trusted Extensions と連動するように変更されています。概念的な情報については、第 5 章「プロセス間通信」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については、第 4 章「印刷とラベル API」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「ラベルビルダー API」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);`

Trusted X Window System API

Trusted X Window System API については、第 6 章「Trusted X Window System」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 Oracle Solaris ライブラリルーチンおよびシステムコール

次の Oracle Solaris インタフェースは Trusted Extensions パラメータを含むか、このガイドで Trusted Extensions インタフェースと一緒に使用されます。

- `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 のシステムコールおよびライブラリルーチン

次の表に、Trusted Extensions のシステムコールおよびルーチンの一覧を示します。この表には、インタフェースの説明および宣言と、このガイドに記載されているインタフェースの例への参照も示されています。マニュアルページのセクションは、各システムコールおよびルーチンの名前の一部として記載されています。

表 B–1 Trusted Extensions で使用されるシステムコールおよび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「ラベルの関係」 40 ページの「ラベルの比較」	48 ページの「2 つのラベル間の関係の特定」
`blequal`(3TSOL)	40 ページの「ラベルの比較」	48 ページの「2 つのラベル間の関係の特定」
`blinrange`(3TSOL)	17 ページの「ラベルの関係」	59 ページの「プリンタのラベル範囲に基づいたラベルリクエストの検証」
`blmaximum`(3TSOL)	40 ページの「ラベルの比較」	
`blminimum`(3TSOL)	40 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および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)	40 ページの「ラベルの比較」	
fgetlabel(2)	29 ページの「ラベル付きゾーン」 35 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「ラベルと文字列の変換」	
getlabel(2)	29 ページの「ラベル付きゾーン」 35 ページの「ファイルのラベルの取得と設定」	46 ページの「ファイルラベルの取得」
getpathbylabel(3TSOL)	37 ページの「ゾーン内のラベルへのアクセス」	
getpeerucred(3C)	55 ページの「get_peer_label() ラベル対応関数」	57 ページの「資格とリモートホストラベルの取得」
getpflags(2)	27 ページの「MAC 除外ソケット」	
getplabel(3TSOL)	34 ページの「プロセス機密ラベルへのアクセス」	84 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)	36 ページの「ラベル範囲の取得」	
getzoneidbylabel(3TSOL)	37 ページの「ゾーン内のラベルへのアクセス」	
getzonelabelbyid(3TSOL)	37 ページの「ゾーン内のラベルへのアクセス」	
getzonelabelbyname(3TSOL)	37 ページの「ゾーン内のラベルへのアクセス」	
getzonerootbyid(3TSOL)	37 ページの「ゾーン内のラベルへのアクセス」	
getzonerootbylabel(3TSOL)	37 ページの「ゾーン内のラベルへのアクセス」	
getzonerootbyname(3TSOL)	37 ページの「ゾーン内のラベルへのアクセス」	
is_system_labeled(3C)	55 ページの「get_peer_label() ラベル対応関数」 33 ページの「Trusted Extensions システムの検出」	56 ページの「印刷サービスがラベル付き環境で実行されているかどうかの確認」
labelbuilder(3TSOL)	第 7 章「ラベルビルダー API」	88 ページの「対話型ユーザーインタフェースの作成」

表 B–1 Trusted Extensions で使用されるシステムコールおよび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
label_to_str(3TSOL)	39 ページの「ラベルと文字列の変換」	45 ページの「プロセスラベルの取得」
m_label_alloc(3TSOL)	35 ページの「ラベル用のメモリーの割り当てと解放」	45 ページの「プロセスラベルの取得」
		46 ページの「ファイルラベルの取得」
m_label_dup(3TSOL)	35 ページの「ラベル用のメモリーの割り当てと解放」	
m_label_free(3TSOL)	35 ページの「ラベル用のメモリーの割り当てと解放」	59 ページの「プリンタのラベル範囲に基づいたラベルリクエストの検証」
		45 ページの「プロセスラベルの取得」
setflabel(3TSOL)	35 ページの「ファイルのラベルの取得と設定」	
	35 ページの「ファイルのラベルの取得と設定」	
setpflags(2)	27 ページの「MAC 除外ソケット」	
str_to_label(3TSOL)	39 ページの「ラベルと文字列の変換」	59 ページの「プリンタのラベル範囲に基づいたラベルリクエストの検証」
		46 ページの「ファイルラベルの取得」
tsol_getrhtype(3TSOL)	38 ページの「リモート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)	78 ページの「属性へのアクセス」	
XTSOLgetPropAttributes(3XTSOL)	78 ページの「属性へのアクセス」	
XTSOLgetPropLabel(3XTSOL)	79 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)	79 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)	83 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および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)	84 ページの「ウィンドウラベルの取得」	
XTSOLgetResUID(3XTSOL)	85 ページの「ウィンドウのユーザー ID の取得」	
	79 ページの「ウィンドウのユーザー ID へのアクセスと設定」	
XTSOLgetSSHeight(3XTSOL)	81 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)	80 ページの「ワークステーションの所有者 ID へのアクセスと設定」	
XTSOLIsWindowTrusted(3XTSOL)	81 ページの「トラステッドパスウィンドウでの作業」	
XTSOLMakeTPWindow(3XTSOL)	81 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「Trusted X Window System」	
XTSOLsetPropLabel(3XTSOL)	79 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)	79 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)	85 ページの「ウィンドウラベルの設定」	
XTSOLsetResUID(3XTSOL)	79 ページの「ウィンドウのユーザー ID へのアクセスと設定」	
XTSOLsetSessionHI(3XTSOL)	80 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)	80 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)	81 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)	80 ページの「ワークステーションの所有者 ID へのアクセスと設定」	

索引

S

T

U

W

X

Z

あ