2006.03.16 第1回LIDS勉強会

TOMOYO Linux

～導入編～

TOMOYO Linuxを導入するための手順の概略について紹介します。

導入手順は？

（１）カーネルのコンパイル

- www.kernel.org からソースをダウンロードする
- TOMOYO Linux パッチを適用する
- カーネルをコンパイルする

（２）ドメインポリシーの作成

- 必要なアクセス許可を「学習モード」で学習させる
- テンポラリファイルとして使用されるパス名を把握する

（３）例外ポリシーのチューニング

- テンポラリファイルのパス名をパターン化する

導入手順は？

（４）ドメインポリシーの再作成

- 必要なアクセス許可を「学習モード」で学習させる

（５）ドメインポリシーのチューニング

- ドメイン単位でパターン化を行う
- アクセス許可の不足が無いかを「確認モード」で確認する

（６）システムの本番稼動

- 「強制モード」で動作させる

2006.03.16 **第1回**LIDS**勉強会**

TOMOYO Linux

～デモ編～

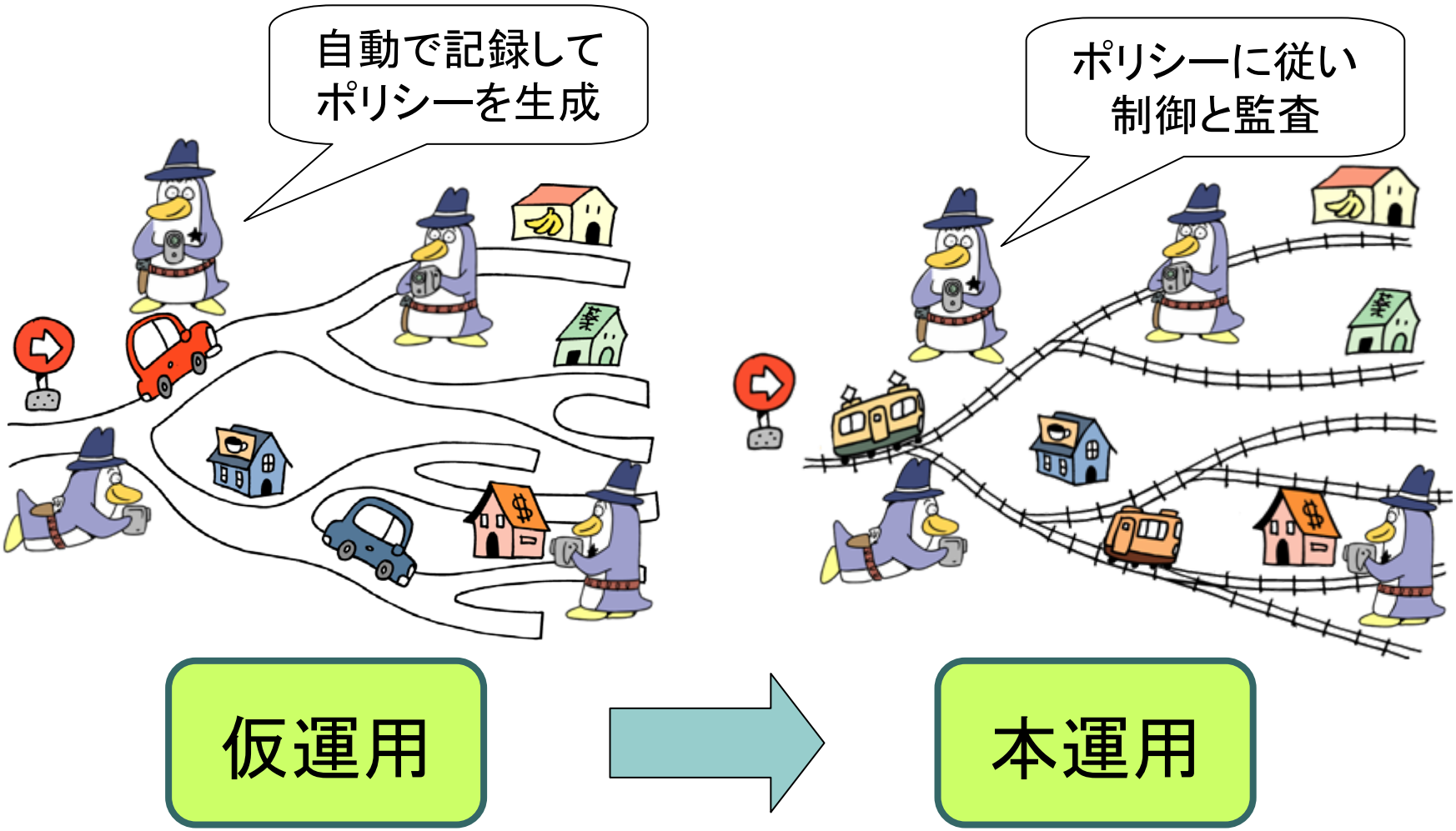
TOMOYO Linuxが得意とする「アクセス許可の自動学習」と、TOMOYO Linuxの構造に由来する「なりすまし対策」について紹介します。

自動学習機能

- TOMOYO Linux は、強制アクセス制御を行うために必要なポリシーを自動的に学習する機能を備えている。

- 従来のセキュリティ強化Linuxでは不可避であった「ポリシーを策定するためには膨大な労力が必要」という難題を解決した。

自動学習機能



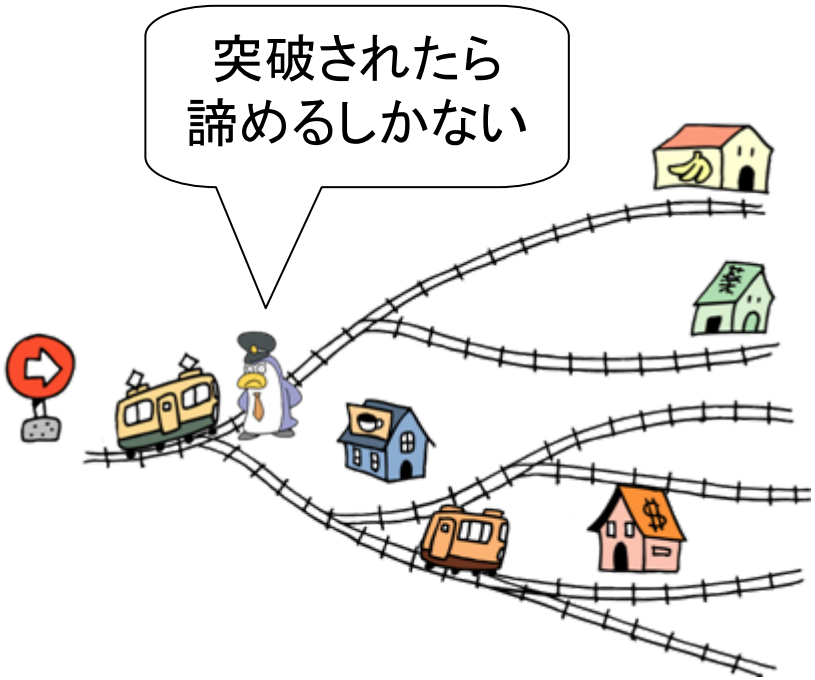
自動学習機能のデモ

- VMware 上の Debian Sarge で、「学習モード」で動作させることでポリシーを生成し、「強制モード」で動作させることでポリシーで許可されていない操作が禁止されていることを示す。

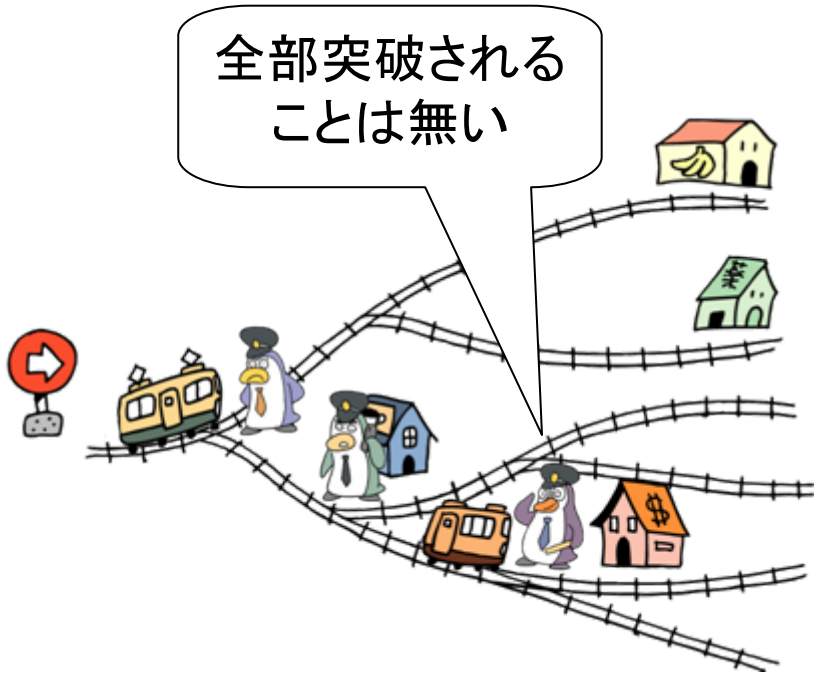
ログイン認証の強化

- 従来の常識では、「ログイン認証は１回限り。だから、突破されたら諦めろ。」
- しかし、強制アクセス制御を使えば、「ログイン認証は何回でも強制可能。だから、諦めることは無い。」
- SecurIDやSecure Matrixのような商用製品を使わないで安全確実なsshログイン認証を実現。コストは限りなく0。

ログイン認証の強化



標準



強化後

ログイン認証強化のデモ

- rootとしてパスワードを使用してsshログインしただけでは何もできないことを示す。
- ログイン後だからこそ可能になる様々な認証方式を使用して追加のログイン認証を行えることを示す。

2006.03.16 第1回LIDS勉強会

TOMOYO Linux

～技術編～

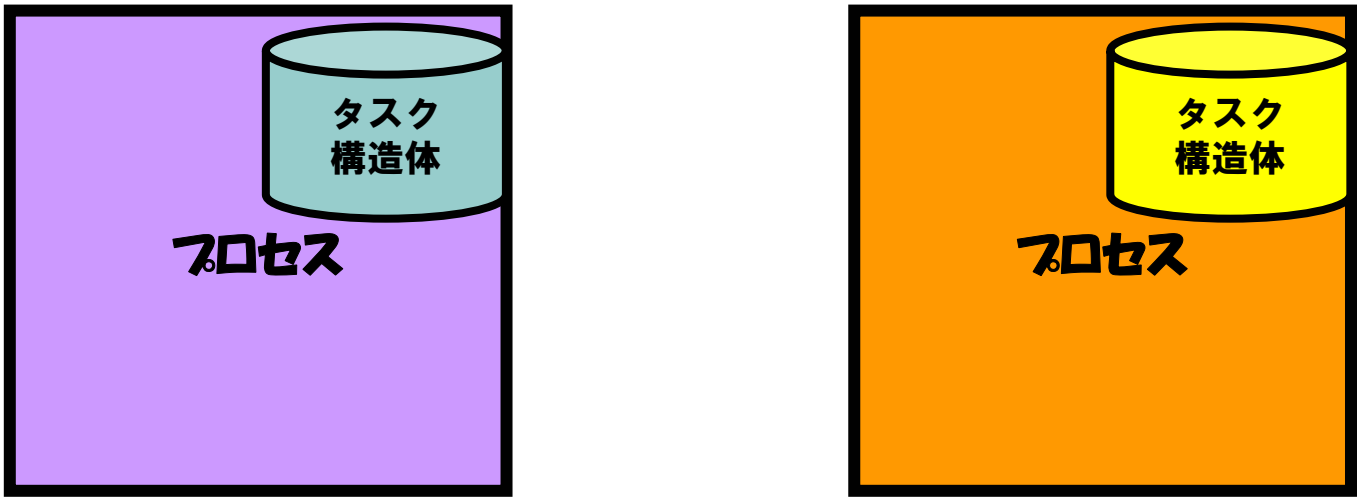
「TOMOYO Linuxは何故アクセス許可を自動的に学習することができるのか」「TOMOYO Linuxは何故システムの振る舞いに沿って忠実に制御できるのか」について紹介します。

アクセス制御の要

- どうすれば「アクセスする側」を詳細に記述できるか？
  - TOMOYO Linux では、「タスク構造体」を活用
- どうすれば「アクセスされる側」を詳細に記述できるか？
  - TOMOYO Linux では、「パス名」を活用

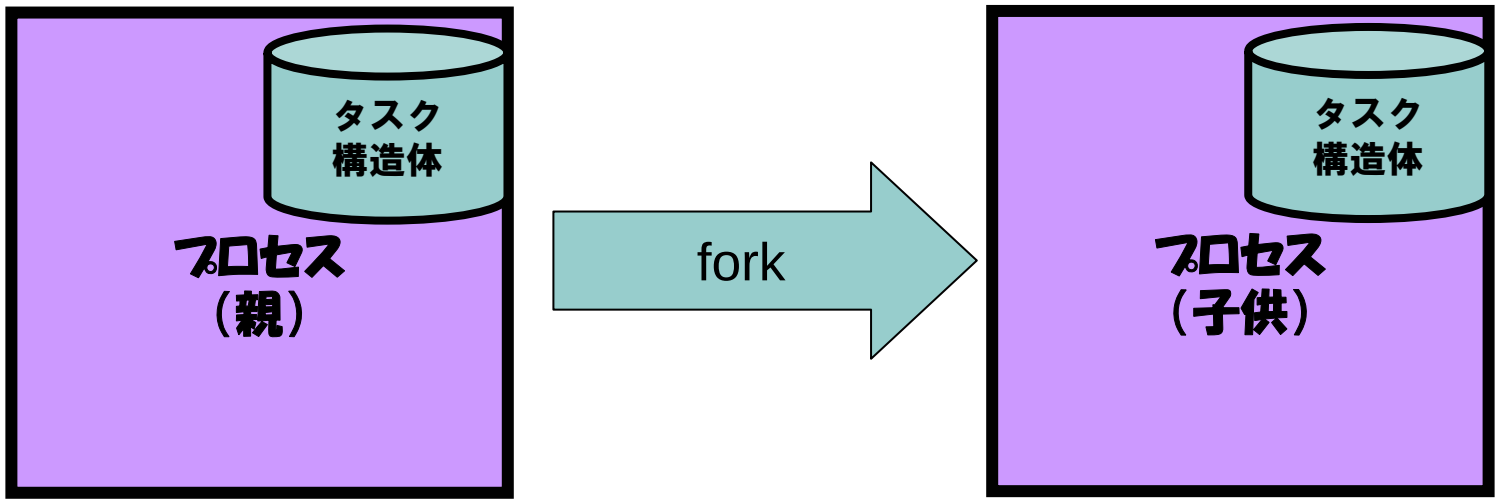
タスク構造体って何？

- タスク構造体（task_struct）とは
  - カーネルがプロセスを管理するために使用するデータベース。



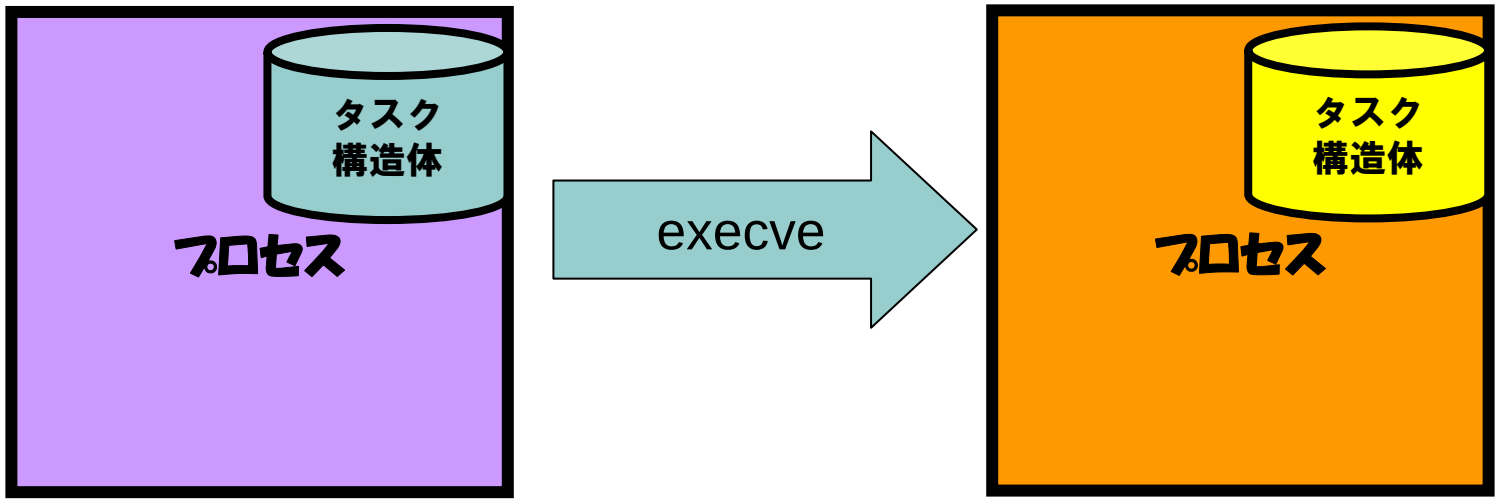
タスク構造体って何？

- タスク構造体（task_struct）とは
  - プロセスを複製するとタスク構造体も複製される。



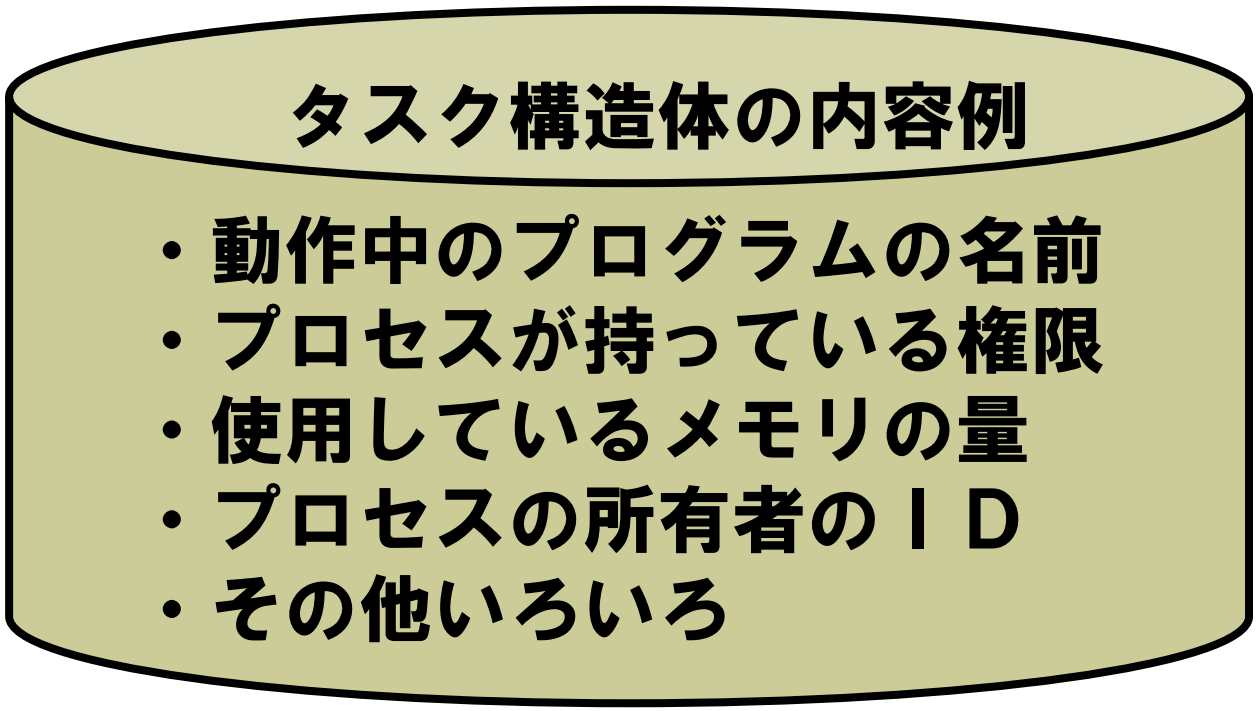
タスク構造体って何？

- タスク構造体（task_struct）とは
  - プログラムを実行するとタスク構造体も更新される。



タスク構造体の利点って？

- 全てのプロセスが持っている。
- 遺伝子のように時系列に沿って情報が伝播する。



タスク構造体の利点って？

- タスク構造体に情報を追加すると、その情報も伝播する。
  - TOMOYO Linux はタスク構造体の利点に注目し、情報を追加した。

何を追加したの？

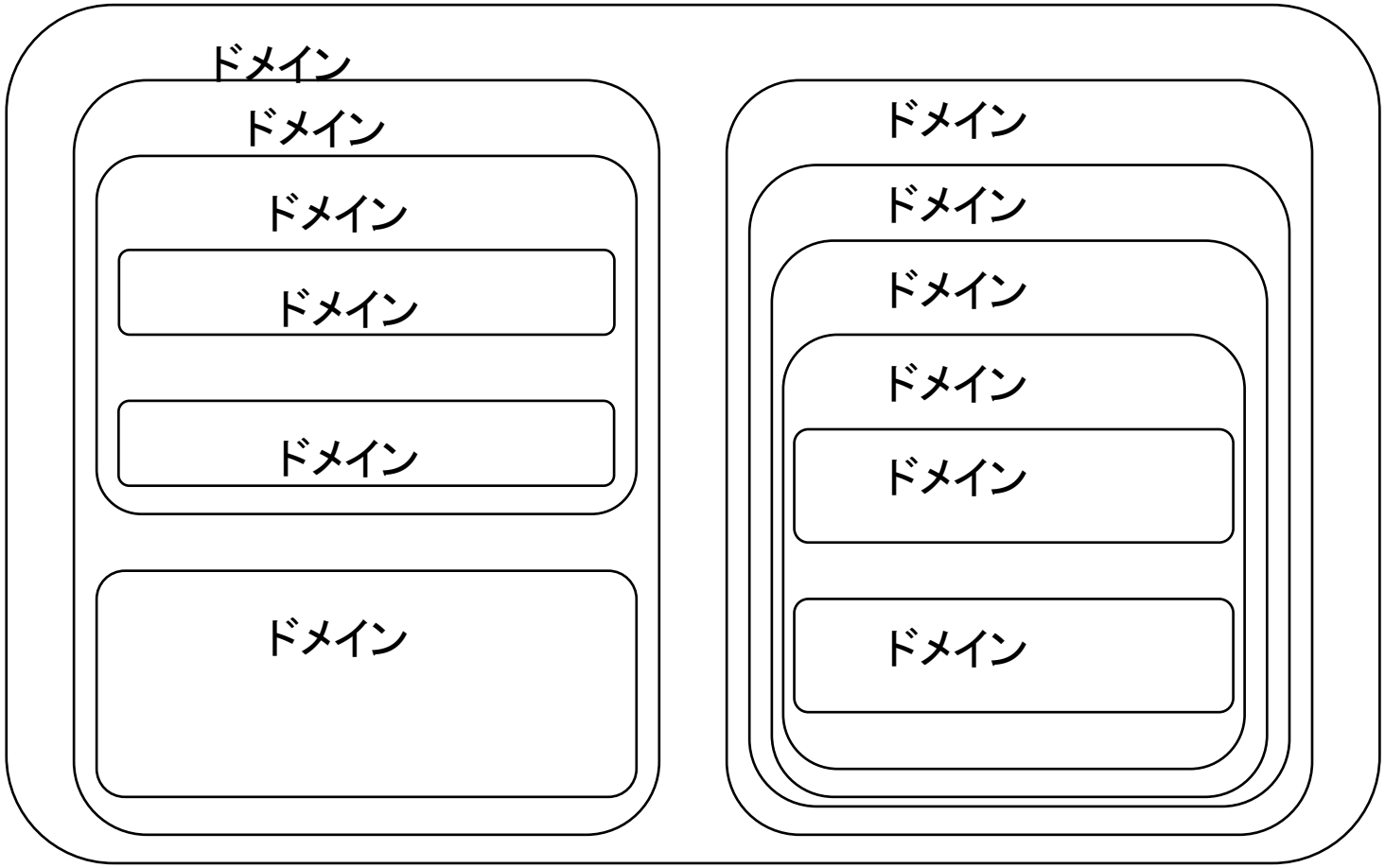
- プログラムの起動履歴
  - 現在のプロセスに至るまでに実行されたプログラムのパス名を保持させるようにした。
  - これにより、時系列によるプロセスの識別が可能になった。
  - プログラムの起動履歴により識別される状態をドメインに対応させることで、ドメイン単位のアクセス制御が可能になった。

ドメインって何？

- プロセスに与えられるアクセス許可を制限するために作成される「グループ」のこと。
- ある「グループ」に属するプロセスは、その「グループ」に対して与えられたアクセスだけが許可される。
- ドメインには名前が付けられる。
  - TOMOYO Linux のドメイン名は、プログラムの起動履歴と同じ
  - SELinux のドメイン名は利用者が定義する

ドメインの構造は？

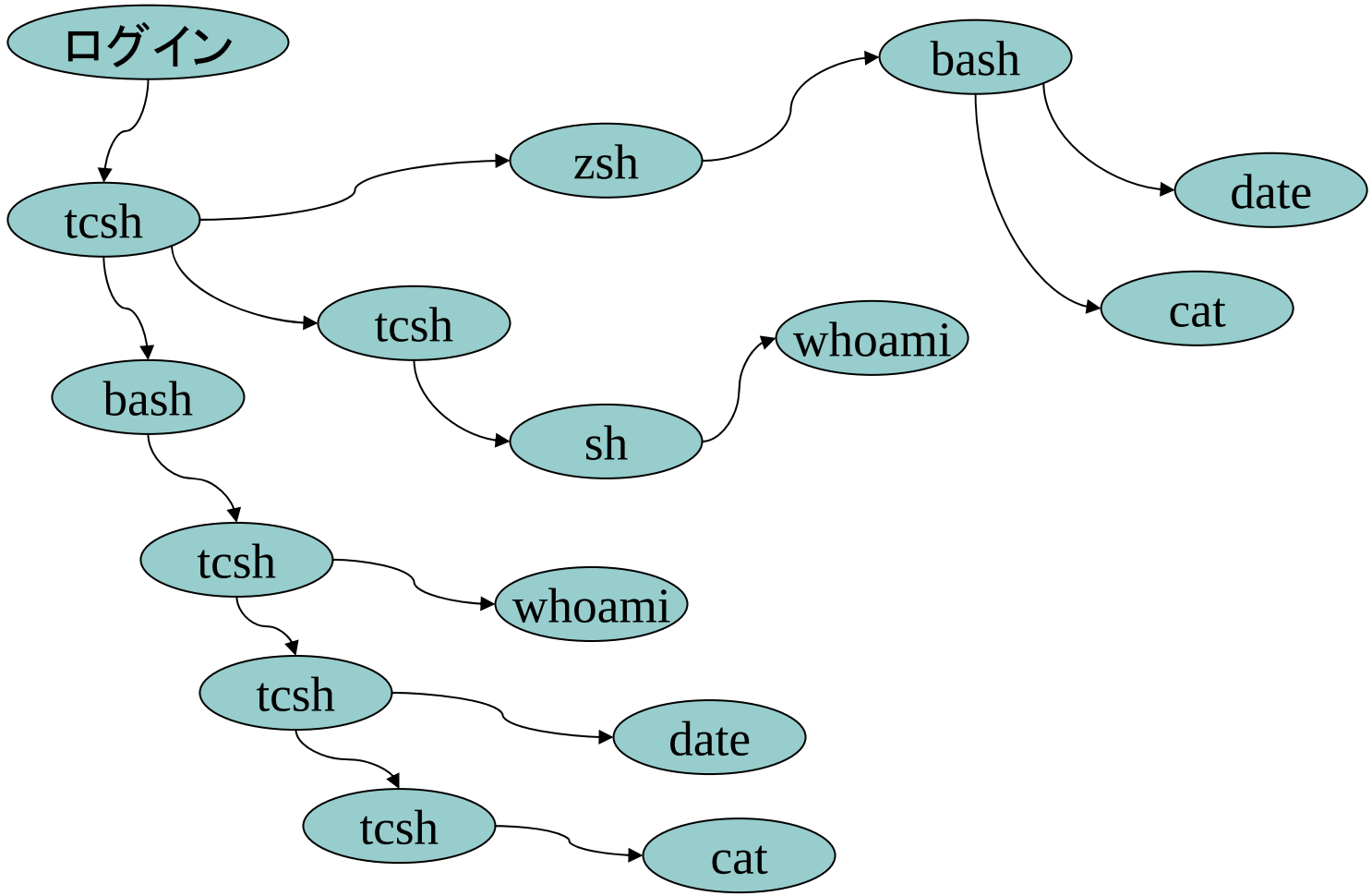
- **階層構造　（SELinux では平坦構造）**



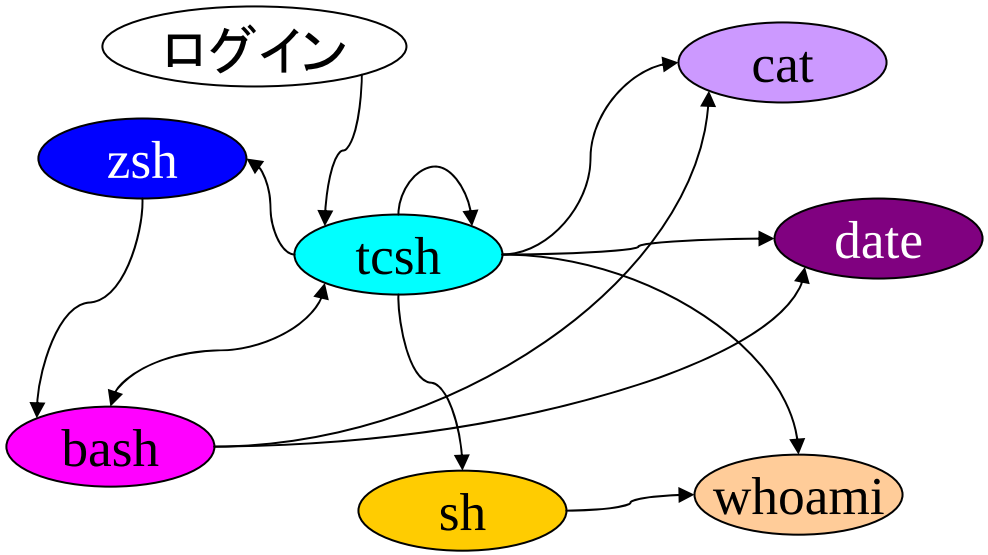
階層構造の利点は？

- 実際のシステムの動作に沿って記述できる。
- 設定変更時の影響範囲を最小化できる。

＜要求例＞以下の許可を付与して欲しい。
必要以上の許可はなるべく付与しないこと。



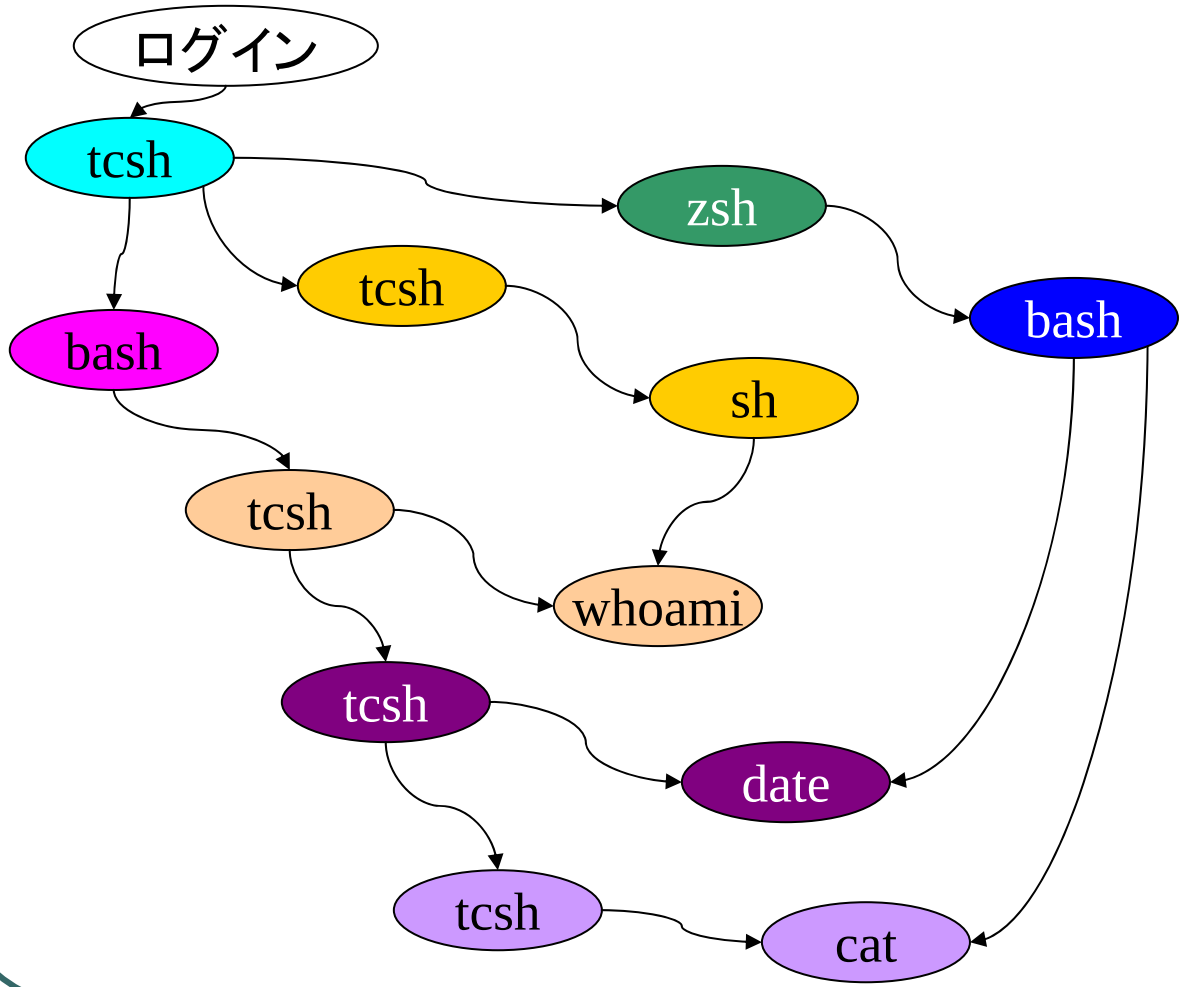
＜設定例1＞余計な許可を容認する場合
（現在のドメインを基点とするアクセス制御方式）



必要なドメインの数は少ないが、循環によって余計な許可を与えている。

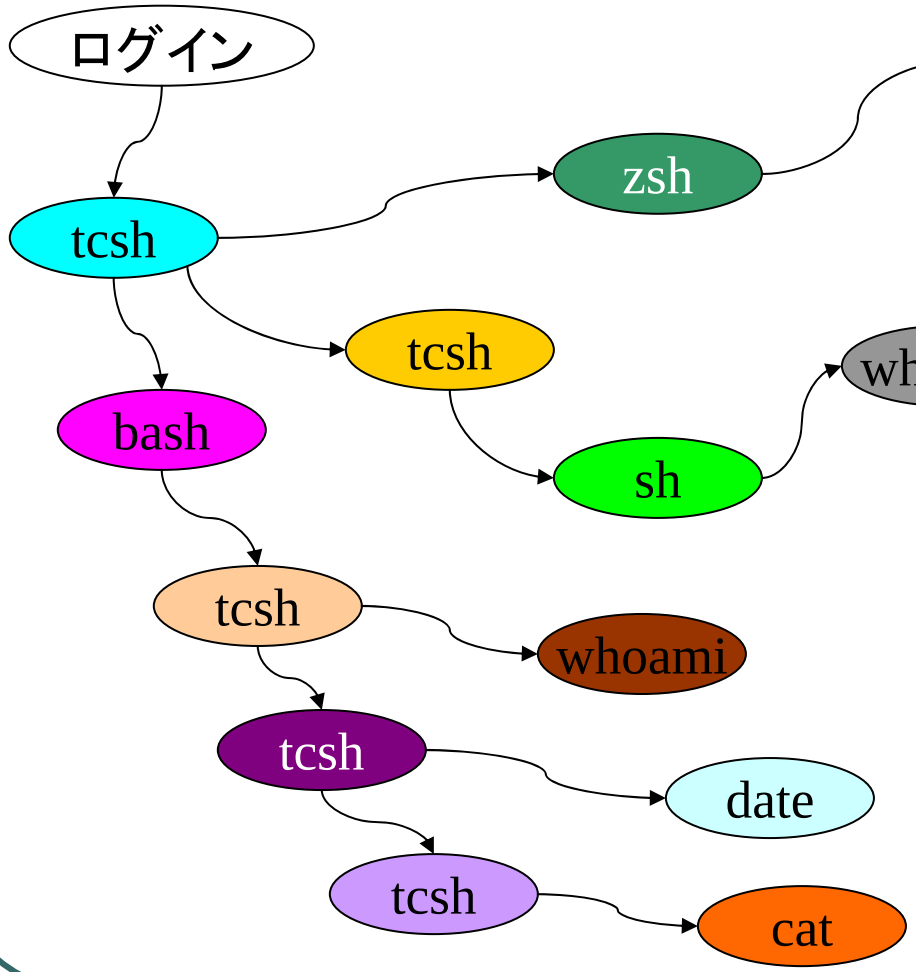
- tcsh から bash , sh , tcsh , zsh , cat , date , whoami を許可
- bash から tcsh , cat , date を許可
- sh から whoami を許可
- zsh から bash を許可 ⇒最低限必要な条件は満たしている。

＜設定例2＞余計な許可を容認できない場合
（現在のドメインを基点とするアクセス制御方式）



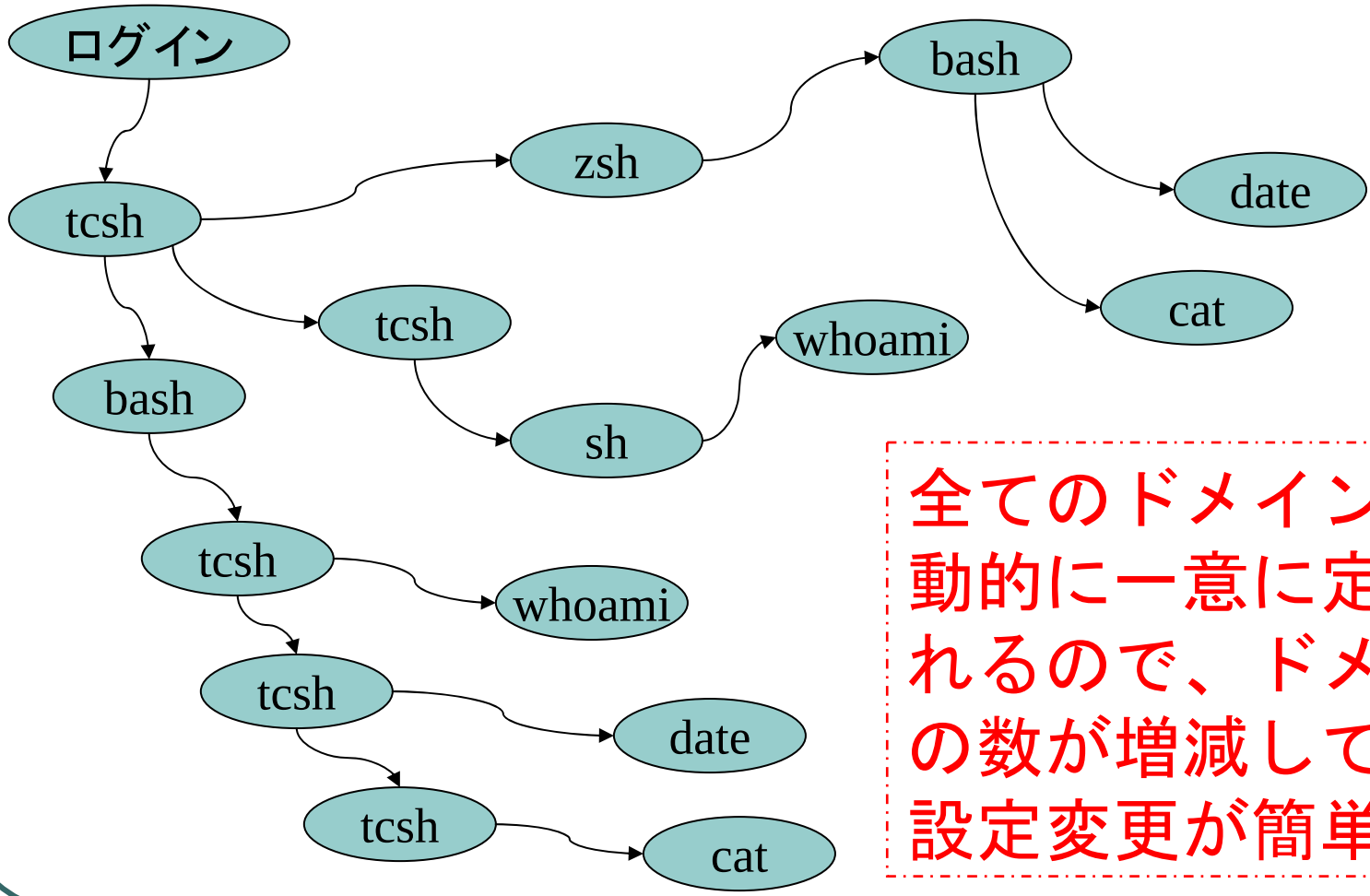
循環が生じないため余計な許可を与えず、必要なドメインの数も少ないが、設定変更による影響範囲に注意が必要。

＜設定例3＞余計な許可を容認できない場合
（現在のドメインを基点とするアクセス制御方式）



設定変更による影響範囲を最小化できるが、ドメインの一意性を保証するための手間が面倒。

＜設定例4＞余計な許可を容認できない場合
（ログインを基点とするアクセス制御方式）



全てのドメインが自動的に一意に定義されるので、ドメインの数が増減しても、設定変更が簡単♪

アクセス対象の指定方法は？

- 絶対パス名で指定
- 以下の4種類のワイルドカードも使用可

  ¥$　　1桁以上の整数

  ¥+　　整数1桁

  ¥*　　/ の直前までの0文字以上

  ¥?　　/ 以外の1文字

- 「サブディレクトリ以下全部」を指定する方法は無い

  「/¥*/¥*」「/¥*/¥*/¥*」のようにする。

パス名の導出方法は？

- プロセスの名前空間の“/”まで遡る
  - chroot 環境下でも正しいパス名を導出
    （AppArmor は chroot の“/”までしか遡らない）
- パス名は ASCII の図形文字のみで表記
  - ASCII の図形文字以外は ¥ooo 形式の8進数に変換
  - ポリシーファイルは「スペース」と「改行」によって区切られた ASCII テキスト形式

BusyBox への対応は？

- ハードリンクを作成してもらうだけで対応可能
  - TOMOYO Linux はシンボリックリンクを解決したパス名で制御するので、シンボリックリンクで区別することはできない
- ラッパープログラムは不要
  - ただし、ハードリンクを作成できないファイルシステムの場合にはラッパープログラムを使う
  - LIDS ではラッパープログラムを使用

セキュリティラベルは？

- 扱わない
  - 情報フロー分析はできない
    - 被害を軽減するための保険となることを目指す
  - 利用者に「iノード」を意識させたくない
    - ハードリンクの問題
  - ラベル割り当て問題を回避するため
    - 両方のラベルへのアクセスを許可したら与えすぎ？
    - ラベルを割り当てなおした場合の影響範囲は？

  （SELinuxでは「情報フロー分析」を行って判断）

アクセス許可の粒度は？

- 基本は「読み」「書き」「実行」の3種類

(--X) プログラムを実行する
(-W-) ファイルを書き込みモードで開く
(-WX) ファイルを書き込みモードで開く／プログラムを実行する
(R--) ファイルを読み込みモードで開く
(R-X) ファイルを読み込みモードで開く／プログラムを実行する
(RW-) ファイルを読み書きモードで開く
(RWX) ファイルを読み書きモードで開く／プログラムを実行する

アクセス許可の粒度は？

- **詳細な書き込みアクセスのチェックも可能**
  （次期バージョンで対応予定）
  - 通常ファイルの作成
  - FIFO の作成
  - UNIX ドメインソケットの作成
  - ブロック型デバイスファイルの作成
  - キャラクタ型デバイスファイルの作成
  - シンボリックリンクの作成
  - ハードリンクの作成
  - ディレクトリ以外の削除
  - ディレクトリの作成
  - ディレクトリの削除
  - ファイルの切り詰めと伸長
  - ファイル名の変更

ディレクトリのアクセス許可は？

- 参照と移動は常に許可
  - 最終的にファイルに対するアクセスが禁止されていれば、途中のディレクトリに対するアクセスが許可されようが拒否されようが結果には影響しない。だから、「ファイル名やディレクトリ名を隠さなければいけない場合」以外は、ディレクトリの参照と移動を許可しても大きな問題にはならない。
  - どちらが常識的な使い方？
    - echo MyPassword > ~/secret/password_file
    - touch ~/secret/password_dir/MyPassword

ディレクトリのアクセス許可は？

- ディレクトリ自身の作成・削除のみチェック
  - 最終的にファイルを作成・削除する権限が無ければ、途中のディレクトリに対するアクセスが許可されようが拒否されようが結果には影響しない。
    だから、ディレクトリに対して、「そのディレクトリ内にファイルやサブディレクトリを作成・削除する」権限をチェックしなくても大きな問題にはならない。
- 例外を積み重ねるような指定をさせない
  - 常に１回の指定でアクセスの可否を判断できるようになっている。

ポリシーの種類は？

- Strict Policy
  - /sbin/init の開始時から電源が切れるまでに実行される全プログラムを保護
- Targeted Policy
  - 特定のプログラムおよびそこから起動されるプログラムのみを保護
  - Strict Policy を策定する自信が無い場合は、Targeted Policy でも大丈夫

デフォルトポリシーは？

- 存在しない
  - 管理者が1から作り上げることができる。
  - だから、デフォルトポリシーが提供されない独自のアプリケーションにも対応できる。
  - 特に、Webアプリケーションに対して効果的。

メンテナンスは？

- 特定のドメイン以下に対しては強制アクセス制御の適用を免除させることができる。
  - ポリシーの編集用
  - ソフトウェアのアップデート用
  - 管理者による非定型業務用

その他には？

- 「ファイルシステムの正しさを保証」

  不正なパーティションがマウントされたら困る。

  ⇒マウントできるパーティションとマウントポイントとファイルシステムの組み合わせを制限できる。

- 「デバイスファイルの正しさを保証」

  不正なデバイスファイルが作成されたら困る。

  ⇒デバイスファイルの名前と属性の対応を強制できる。

その他には？

- 「不正なsshログインを防止」

  管理者として不正にsshログインされては困る。

  ⇒ログイン認証を複数回実行させることができる。

- 「管理者業務の分担も可能」

  複数人で管理できないと困る。

  ⇒ドメイン分割により、管理者業務の一部を委任できる。

まとめ

- システムの状態を詳細に記述できるドメイン
- 直感的に理解できるパス名
- 実測に基づく無駄の無いアクセス許可
- １から構築できる理解可能なポリシー構文