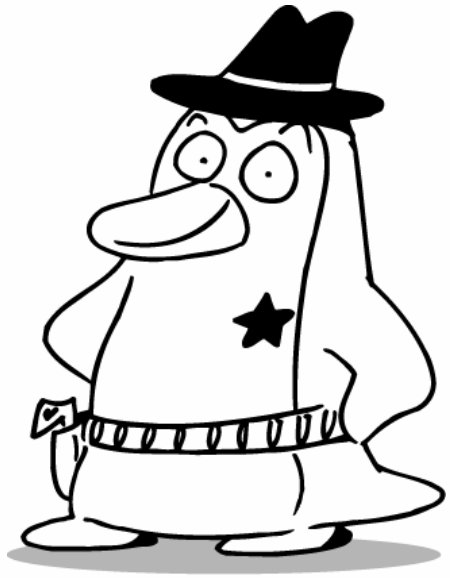
TOMOYO Linux

タスク構造体の拡張によるセキュリティ強化 Linux

平成16年6月3日
株式会社NTTデータ
技術開発本部
オープンシステムアーキテクチャグループ
原田季栄
haradats@nttdata.co.jp



発表内容

- タスク構造体を使用する理由
  - Linuxのプロセス生成の仕組みに注目
- ファイル名を用いたアクセス制御
  - 実装の簡単さ、扱いやすさ
- アクセスポリシーの自動定義機能
  - 管理者の負担を大幅に軽減
- プロセス単位の権限放棄
  - 既存の強制アクセス制御と併用可能な「最小権限」の実装
- プロトタイプシステムの紹介

目次

開発の経緯

- 面倒なポリシーの管理運用なしにLinuxのセキュリティを強化する方法を検討してみた。

↓

- 改ざん防止という目的に限定し、読み込み専用メディアを用いたLinuxサーバの構築に成功した。

↓

- しかし、改ざん防止だけでは不安が残る。
  - 情報漏洩を防止するためには、強制的なアクセス制御が欲しい。
  - でも、面倒なポリシーの管理運用は避けたい。

↓

- どうすればポリシー管理運用の手間を激減できないか？

目標の設定とアプローチ

目標設定

- 十分な管理コストを割けないLinuxサーバを
- ポリシー管理運用の負担を最小限に抑えながら
- 安全、確実に不正アクセスから守る

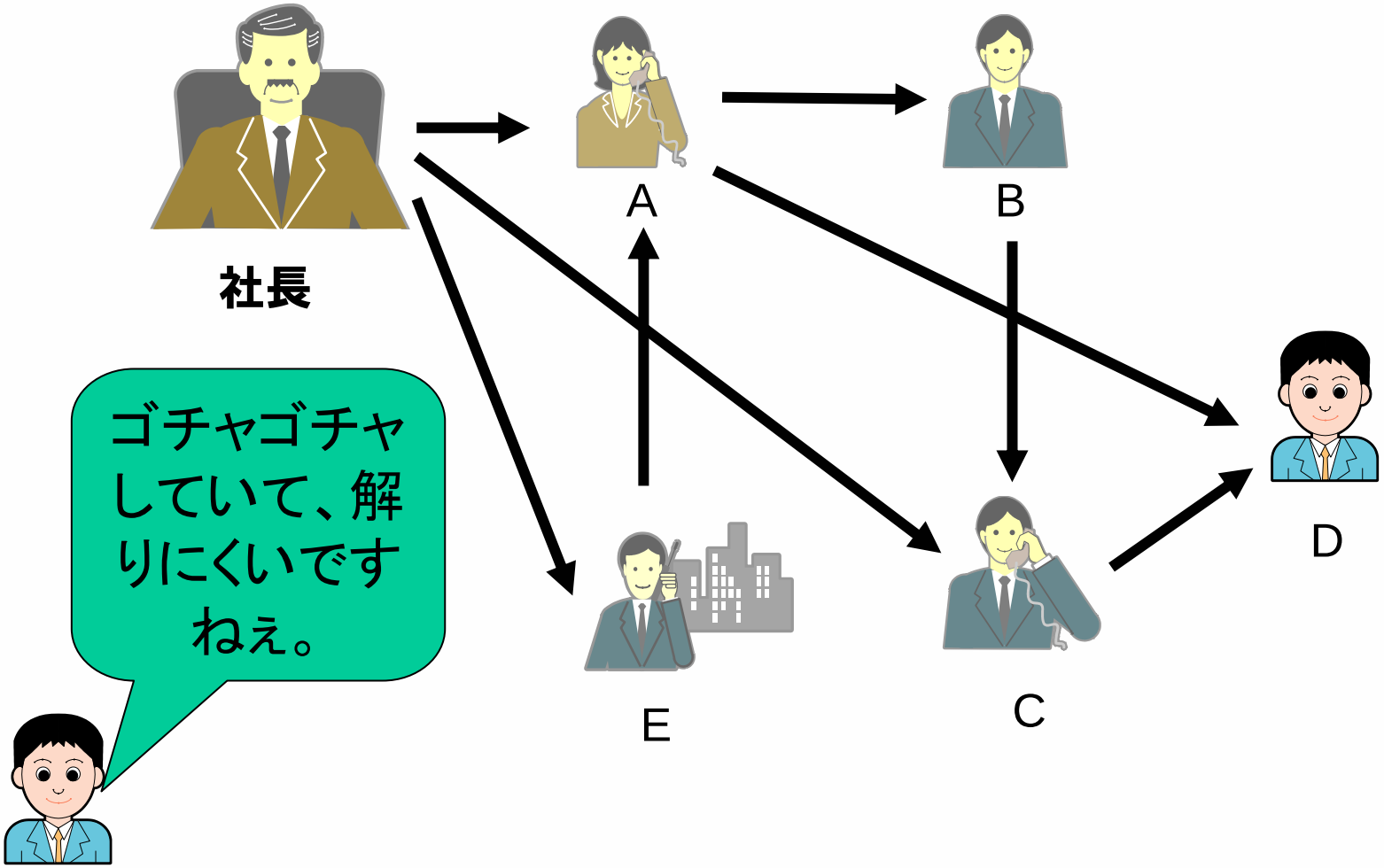
アプローチ

- プロセスに対する強制アクセス制御を中心として
- 管理者が理解しやすく扱いやすい方法で

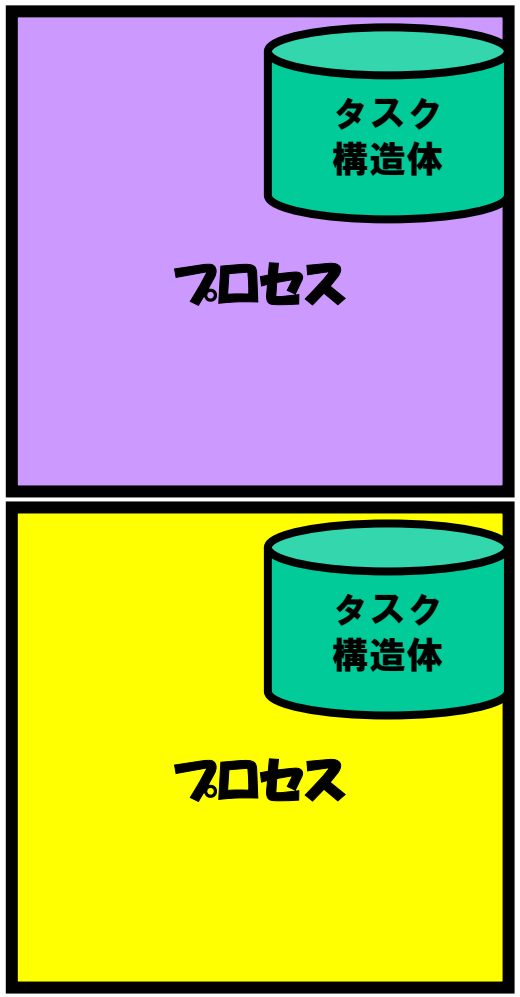
既存の強制アクセス制御の課題

- アクセスポリシーの管理運用が大きな負担
  - Linuxのプロセス生成の仕組みに注目
  - 構文が複雑
  - ドメイン(プロセスの状態)遷移が把握しにくい
    - 次ページの図を参照
  - ファイルに対するラベル付けが必要
  - 不要なアクセス許可を検出する方法が無い
- 回避策
  1. 複雑すぎる条件を使わない
  2. ドメイン遷移をツリー構造にする
  3. ファイル名に基づくアクセス許可の指定
  4. 実測に基づく必要十分なアクセス許可の指定

既存の強制アクセス制御の課題

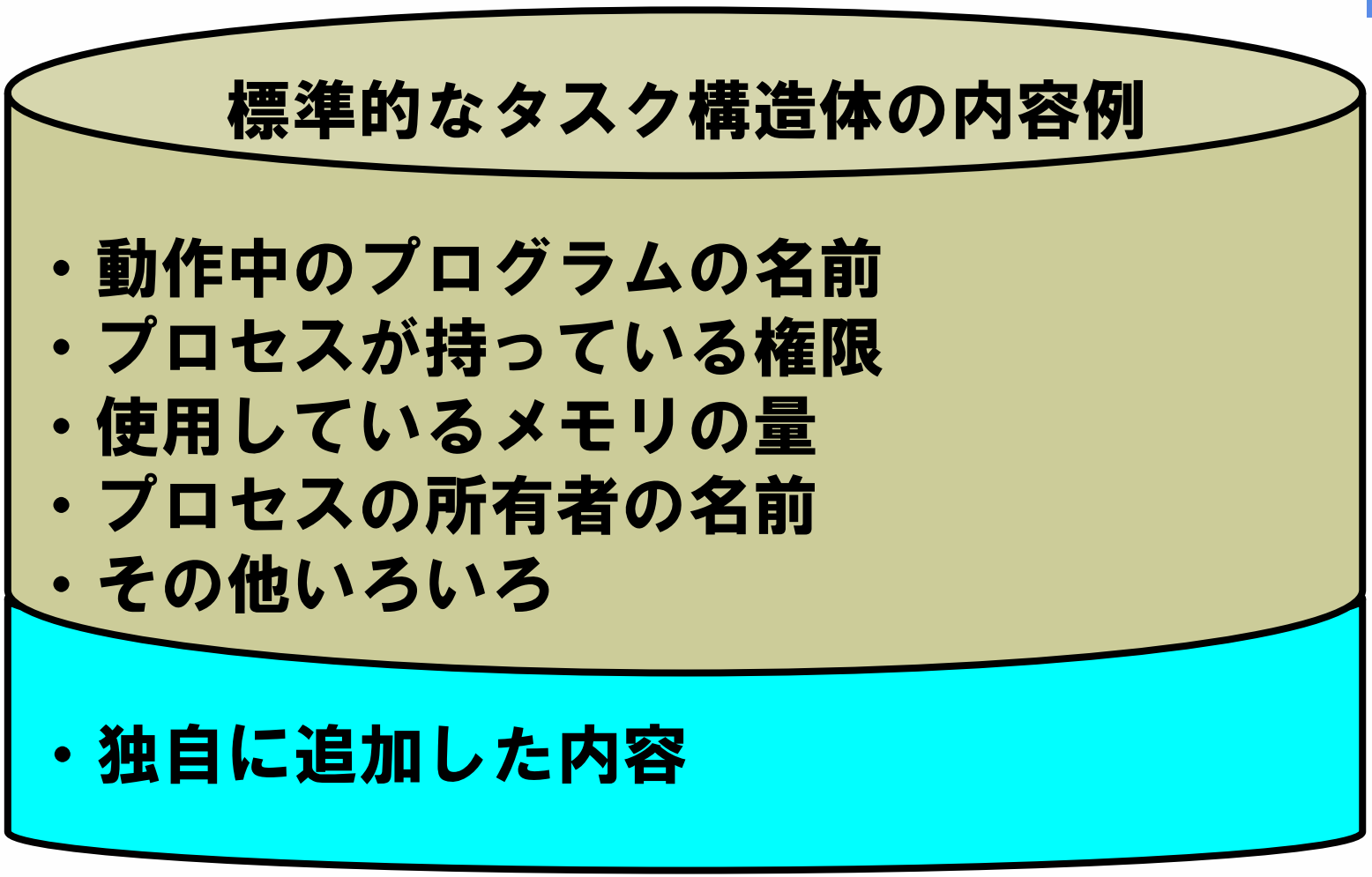


タスク構造体の特徴



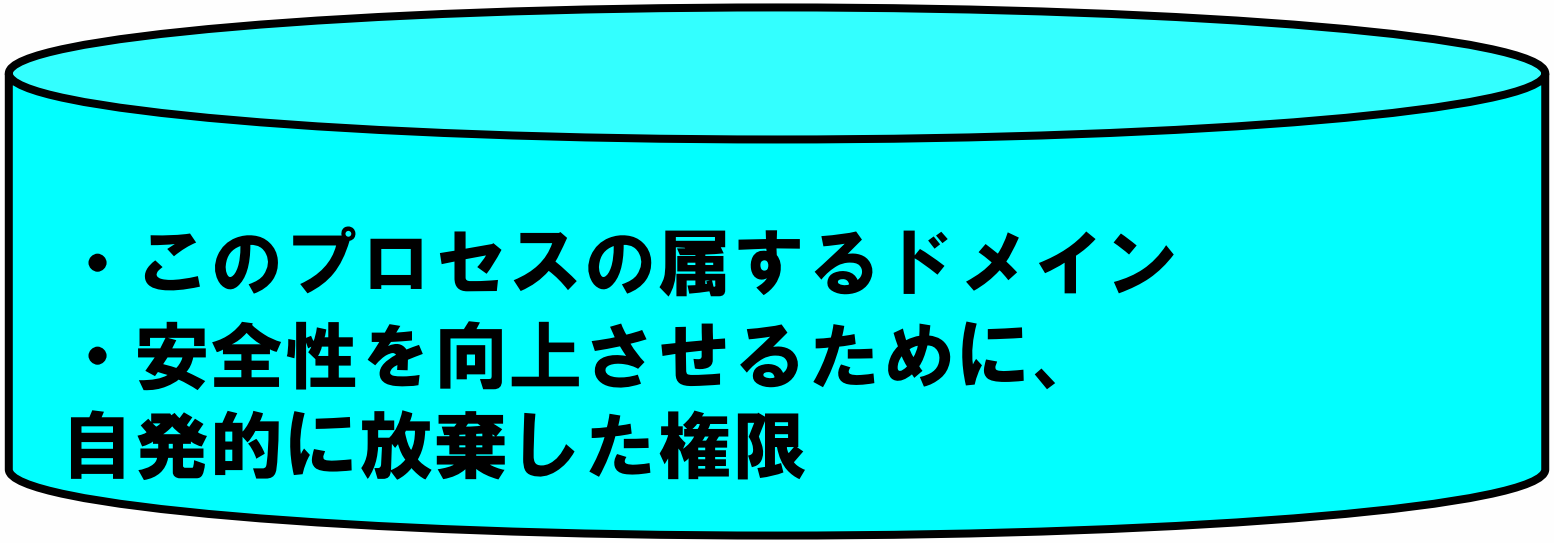
- 各プロセスは「タスク構造体」というデータベースを持っている。
- 「タスク構造体」には、プロセスに関する重要な情報が含まれている。
- 「タスク構造体」に含まれる情報は、カーネルの設計者が自由に追加できる。

タスク構造体の特徴

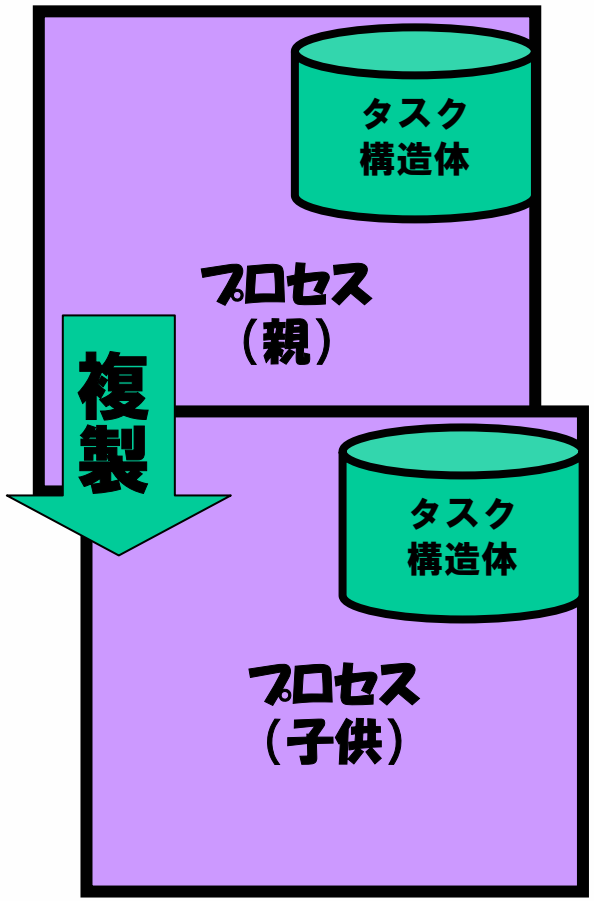


タスク構造体の特徴

・TOMOYO Linuxでは、以下の内容を「タスク構造体」に追加している。



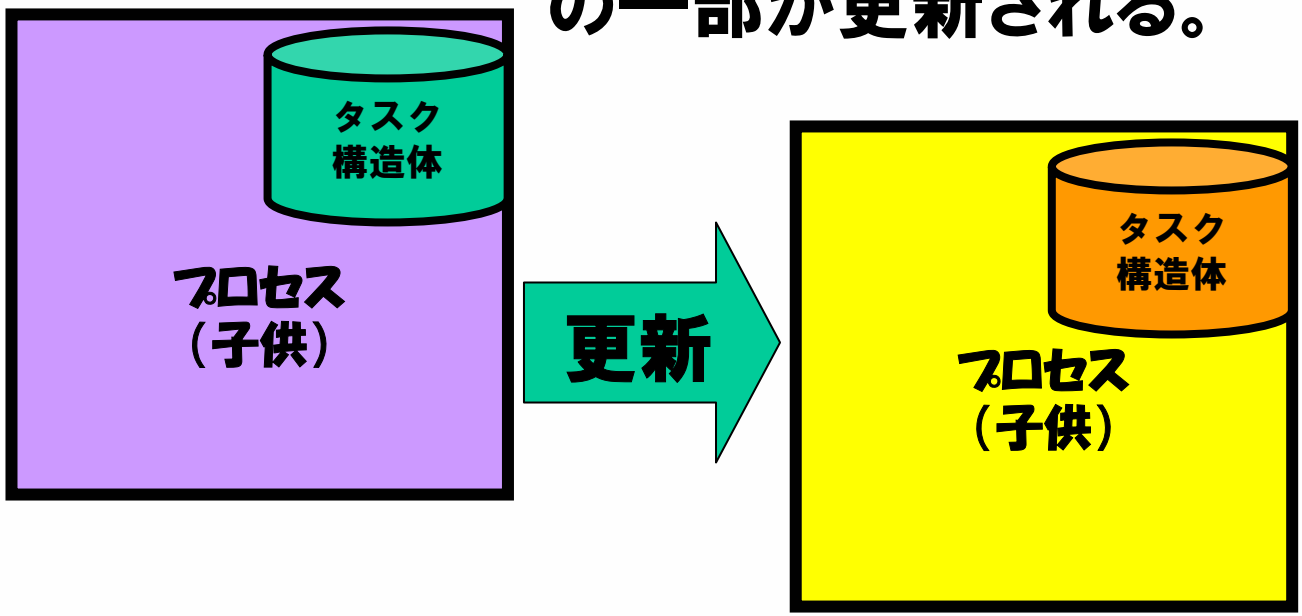
タスク構造体の特徴



- プロセスには親子関係がある。
- 親プロセス（親）は、子プロセス（親のコピー）を作成することができる。
- 子プロセスは親プロセスの「タスク構造体」をそのまま引き継ぐ。（プロセスIDだけ異なる。）

タスク構造体の特徴

- プロセスは別のプログラムを実行することができる。
- このとき、「タスク構造体」の内容の一部が更新される。



TOMOYO Linuxとは

- Task Oriented Management Obviates Your Onus on Linux の略
  - 「タスク指向の管理はLinuxに対するあなたの重荷を予防します」という意味。
- 特徴１：強制アクセス制御
  - アクセスポリシーの自動定義機能
  - アクセスポリシーに基づく強制アクセス制御機能
- 特徴２：自発的アクセス制御
  - アクセスポリシーによらないセキュリティ強化策

TOMOYO Linuxの強制アクセス制御

プロセスに対するアクセス制御

- ユーザに対する役割分担(ロール)の概念は無い
- 構造が単純になる
  - 「どのプロセスが」「どのファイルに」アクセスできるかの条件だけで、多くの用途では十分であると考える

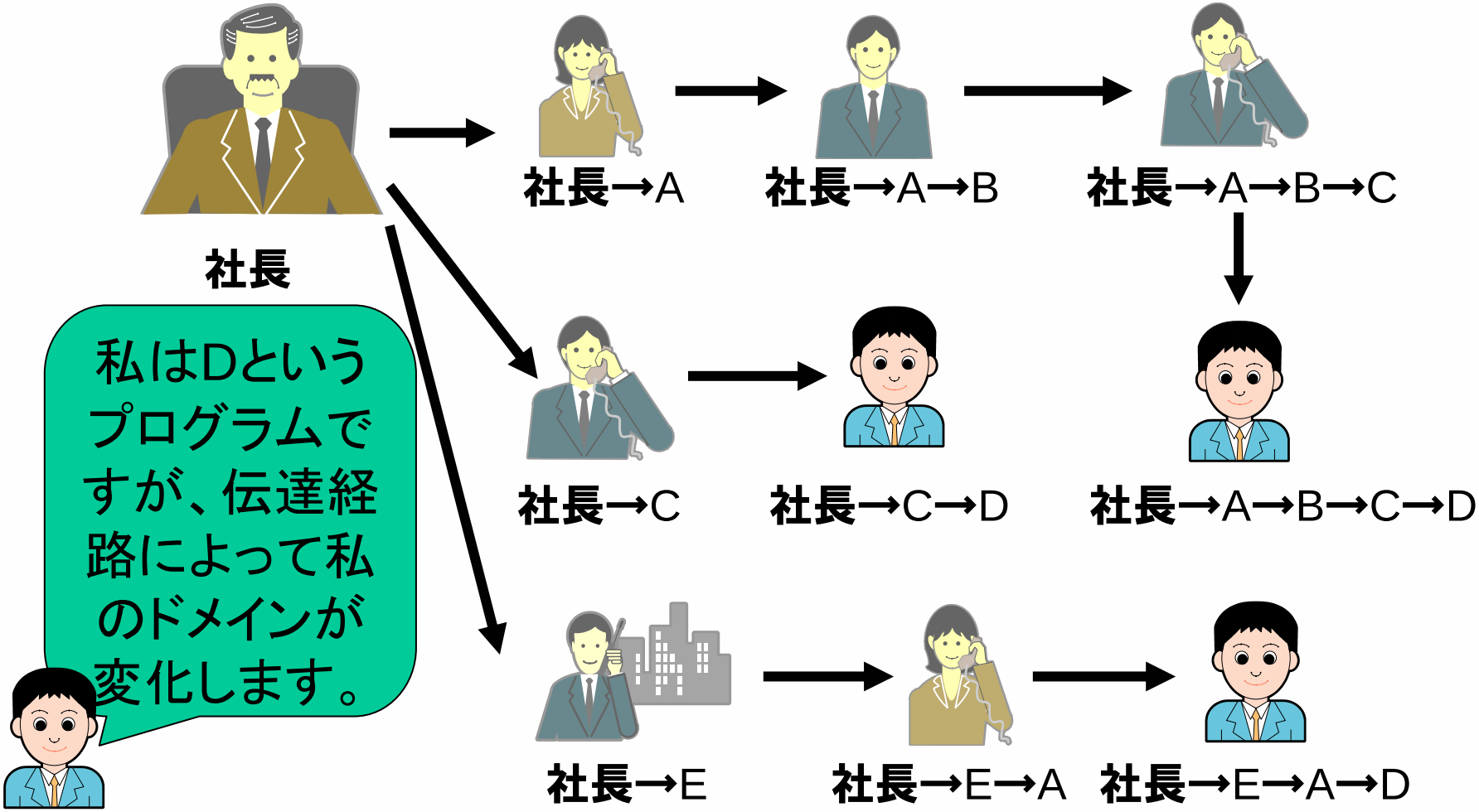
ファイル名によるアクセス制御

- ラベルの概念は無い
  - 正規化された絶対パス名を「ラベル」として使用
- 実装が単純になる
  - 常に正しい「ラベル」が付与されている
  - 「ラベル」の内容を理解しやすい

TOMOYO Linuxの強制アクセス制御

- 独自のドメイン分割
  - 同一プログラムに対しても、親プロセスが1つでも異なれば異なるドメインに属する
    - 次ページの図を参照
  - 構造が単純になる
    - ツリー構造として扱うため、追加や削除が容易
    - 機械的に分割できるので、自動定義が可能
  - 同一プログラムに対して、ドメイン毎に異なるアクセス許可を与えられる
    - 状況に応じた必要最小限のアクセス許可を与えられる

TOMOYO Linuxの強制アクセス制御



TOMOYO Linuxの自発的アクセス制御

- アプリケーションが不要な権限を自発的に放棄することでセキュリティを向上

- アクセスポリシーで許可されたアクセス許可をさらに制限することが可能
  - アプリケーション自身が判断するため、アクセスポリシーが不要
  - 既存のアクセスポリシーに基づく強制アクセス制御との併用が可能

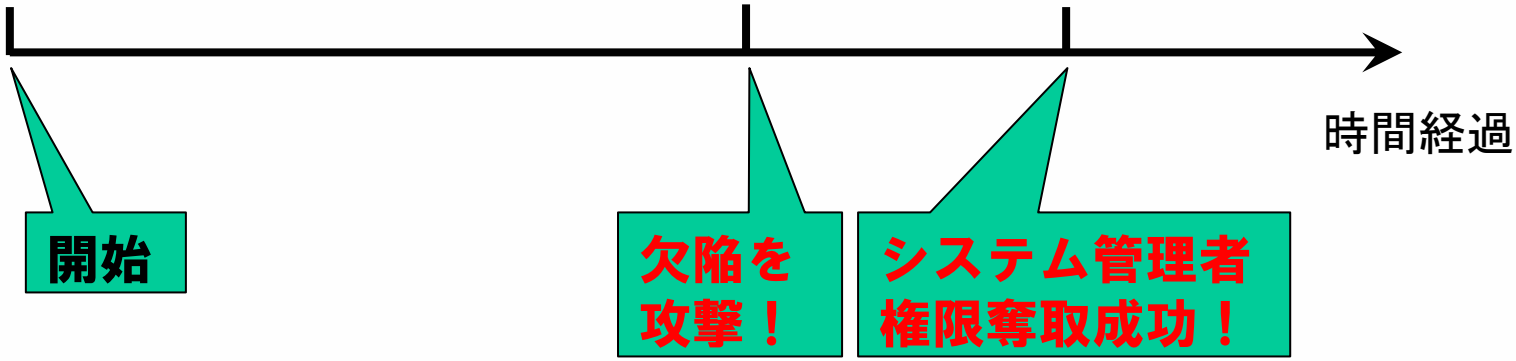
- タスク構造体を使用することで、プロセス単位での制御が可能

TOMOYO Linuxの自発的アクセス制御

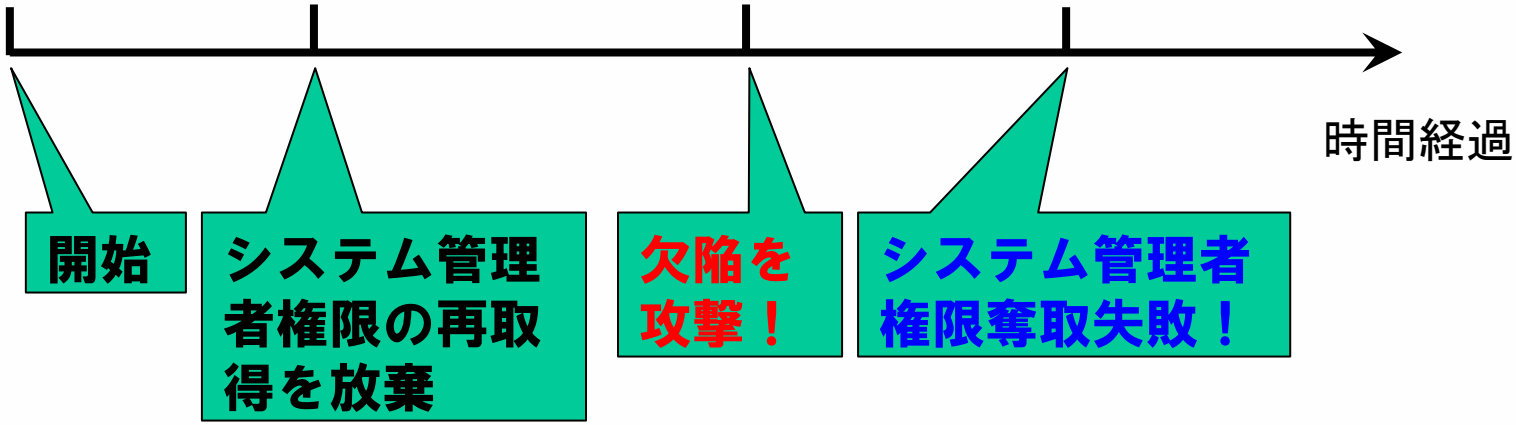
- 一度放棄した権限は再取得不能
- 権限を放棄した後に生成された子プロセスに対しても有効
  - 「タスク構造体」が複製されるときに引き継がれる
- 現時点では以下の権限を放棄可能
  - 新しいプログラムを実行する権限
  - ルートディレクトリを変更・交換する権限
  - マウント操作を実行する権限
  - 一度放棄したシステム管理者権限を再取得する権限

TOMOYO Linuxの自発的アクセス制御

自発的に権限を放棄できない標準的な Linux



自発的に権限を放棄できるTOMOYO Linux



TOMOYO Linuxの自発的アクセス制御

- 応用可能な例(現時点では未実装)
  - TCP/IPネットワークを利用する権限の放棄
  - 子プロセスを生成する権限の放棄
  - TCPポート80を使用できる権限のみの継承
    - タスク構造体に「TCPポート80の使用を許可」というフラグを用意
    - システム管理者権限を持つプロセスがフラグを立てた後、システム管理者権限を持たない子プロセスを実行する。このフラグは子プロセスに引き継がれる
    - システム管理者権限を持たないプロセスでも、「TCPポート80の使用を許可」というフラグが立っていれば許可するようにカーネルを修正する

SAKURA Linuxとは

- Security Advancement Know-how Upon Read-only Approach for Linux の略
  - 「読み込み専用アプローチによるLinuxセキュリティ向上ノウハウ」という意味
  - Linux Conference 2003 にて「読み込み専用マウントによる改ざん防止Linuxサーバの構築」として発表
- 特徴：物理的な改ざん防止
  - CD-ROM 、DVD-ROM 、USBフラッシュメモリ等から起動
  - プログラムや設定ファイルの改ざんを確実に防ぐことができる
  - アクセスポリシーの管理運用が不要
  - RedHat Linux が読み込み専用メディア上で動作する
  - TOMOYO Linuxと同じ「自発的アクセス制御」を備える

プロトタイプシステムの紹介

特長

- 自動定義機能により手間がほとんどかからない
- SAKURA Linux と組み合わせることで「改ざん防止」と「アクセス制御」の両方を実現
- 多数のアプリケーションが動作可能

  Apache 2、LXR、glimpse、Another HTML-lint、
  Tomcat 4、VSFTP、BIND、DHCP、Sendmail、
  Samba、F-Secure AntiVirus for Samba、
  OpenSSH、emacs 等

プロトタイプシステムの紹介

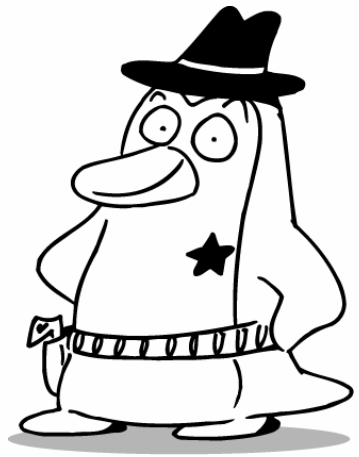
- 運用手順

1. アプリケーションをインストールする
2. （SAKURA Linuxと組み合わせる場合は）読み込み専用化のための修正および設定変更を行う
3. アクセスポリシー自動定義モードで動作させ、許可したい操作を一通り実行する
4. エディタでアクセスポリシーを修正する
5. 強制アクセス制御モードでの運用を開始する

おわりに

- Linux利用者の方々へ
  「TOMOYO Linuxは簡単です。」
  既存のセキュリティ強化Linuxで挫折された方は是非♪

- カーネル開発者の方々へ
  「一緒に開発しませんか？」
  ファイル以外のアクセス制御機能も追加したい。
  「タスク構造体を有効活用しませんか？」
  いろいろ応用できます。

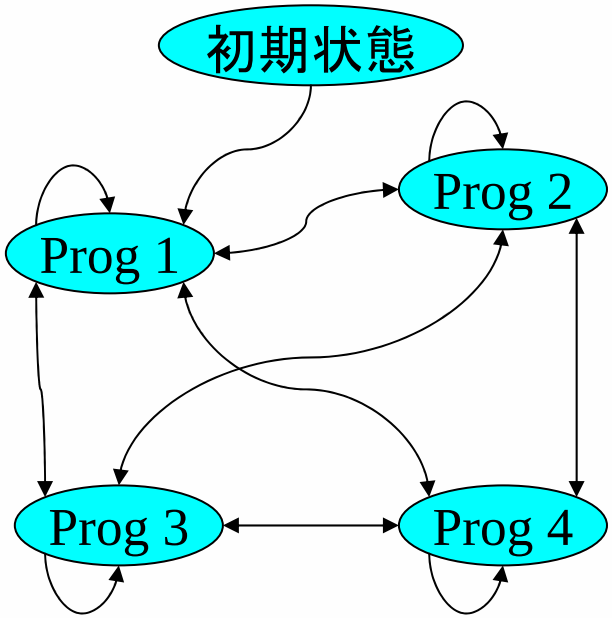


参考文献

- 日経システム構築 2004年4月号 no.132「解説」
  - 「セキュアなシステムを作る(3つの原則に従いOSの機能を強化)」
- 読み込み専用メディア上でのLinuxサーバの運用について
  - 「読み込み専用マウントによる改ざん防止Linuxサーバの構築」
    Linux Conference 2003
    http://lc.linux.or.jp/lc2003/30.html
    原田季栄、保理江高志、田中一男
- 強制アクセス制御のポリシー定義の自動化について
  - 「プロセス実行履歴に基づくアクセスポリシー自動生成システム」
    Network Security Forum 2003
    http://www.jnsa.org/award/2003/result.html
    原田季栄、保理江高志、田中一男
- http://www11.plala.or.jp/tsh/
  - 本資料掲載内容のフォローアップ情報を公開します

補足資料

＜強制アクセス制御の存在しない場合＞

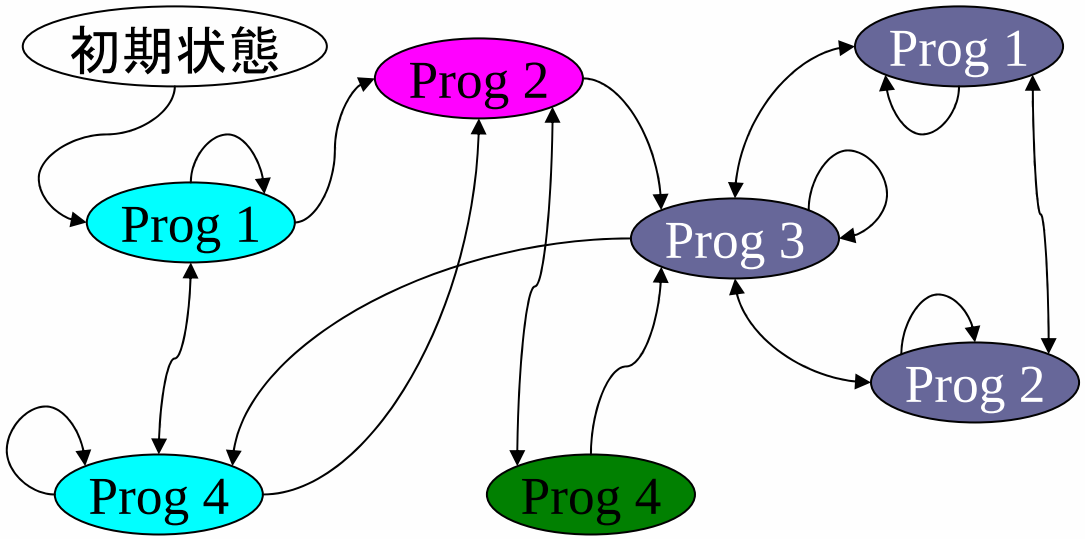


現在の状態	プログラム名	実行の可否	実行後の状態
0	Prog 1	○	Prog 1
Prog 1	Prog 1	○	Prog 1
	Prog 2	○	Prog 2
	Prog 3	○	Prog 3
	Prog 4	○	Prog 4
Prog 2	Prog 1	○	Prog 1
	Prog 2	○	Prog 2
	Prog 3	○	Prog 3
	Prog 4	○	Prog 4
Prog 3	Prog 1	○	Prog 1
	Prog 2	○	Prog 2
	Prog 3	○	Prog 3
	Prog 4	○	Prog 4
Prog 4	Prog 1	○	Prog 1
	Prog 2	○	Prog 2
	Prog 3	○	Prog 3
	Prog 4	○	Prog 4

アクセス許可制限が存在しない

⇒セキュリティ上、問題がある

＜基点を固定しない強制アクセス制御の場合＞



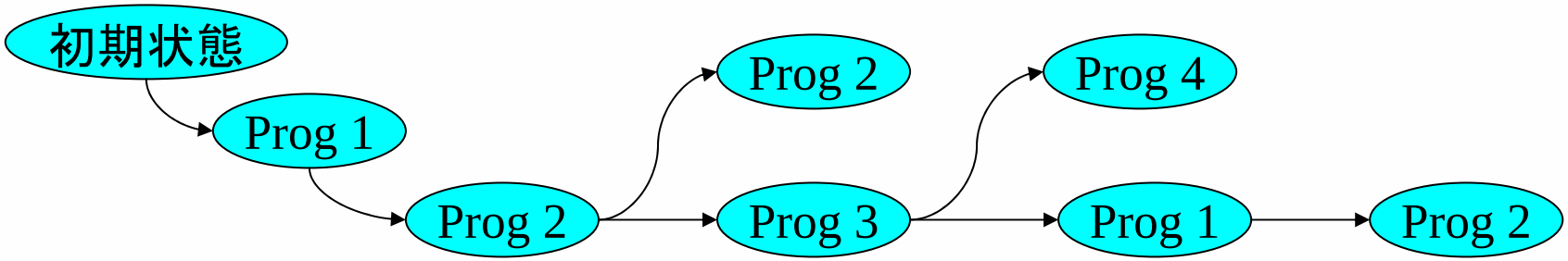
現在の状態	プログラム名	実行の可否	実行後の状態
0	Prog 1	○	A
	Prog 2	×	
	Prog 3	×	
	Prog 4	×	
A	Prog 1	○	A
	Prog 2	○	B
	Prog 3	×	
	Prog 4	○	A
B	Prog 1	×	
	Prog 2	×	
	Prog 3	○	C
	Prog 4	○	D
C	Prog 1	○	C
	Prog 2	○	C
	Prog 3	○	C
	Prog 4	○	A
D	Prog 1	×	
	Prog 2	○	B
	Prog 3	○	C
	Prog 4	×	

許可の一覧表を定義できても流れを追えない

⇒不要な許可を与えていても気が付かない

＜基点を固定した強制アクセス制御の場合＞

現在の状態	プログラム名	実行の可否	実行後の状態
0	Prog 1	○	0→Prog 1
0→Prog 1	Prog 2	○	0→Prog 1→Prog 2
0→Prog 1→Prog 2	Prog 2	○	0→Prog 1→Prog 2→Prog 2
0→Prog 1→Prog 2	Prog 3	○	0→Prog 1→Prog 2→Prog 3
0→Prog 1→Prog 2→Prog 3	Prog 4	○	0→Prog 1→Prog 2→Prog 3→Prog 4
0→Prog 1→Prog 2→Prog 3	Prog 1	○	0→Prog 1→Prog 2→Prog 3→Prog 1
0→Prog 1→Prog 2→Prog 3→Prog 1	Prog 2	○	0→Prog 1→Prog 2→Prog 3→Prog 1→Prog 2

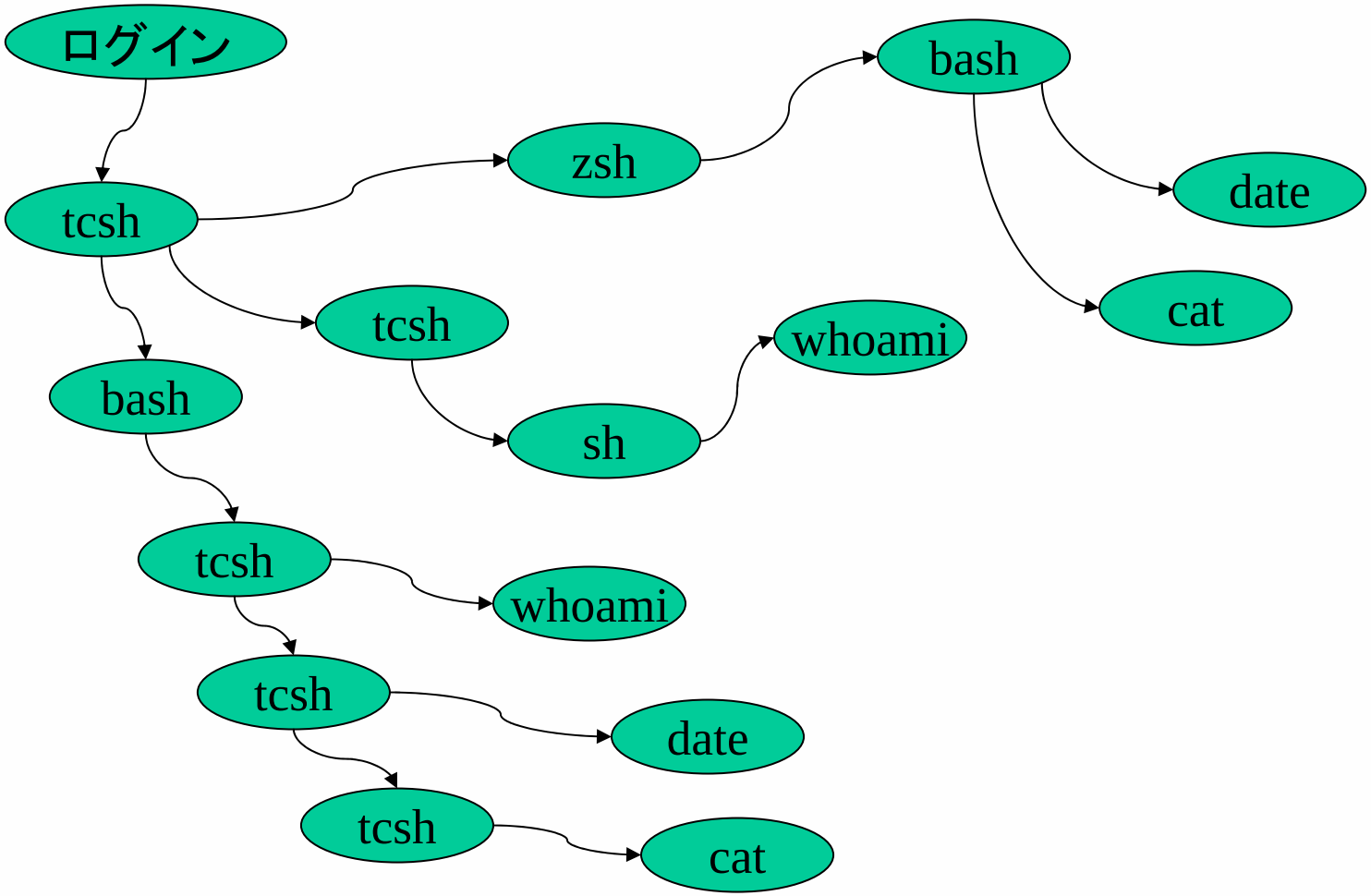


図で理解することが簡単

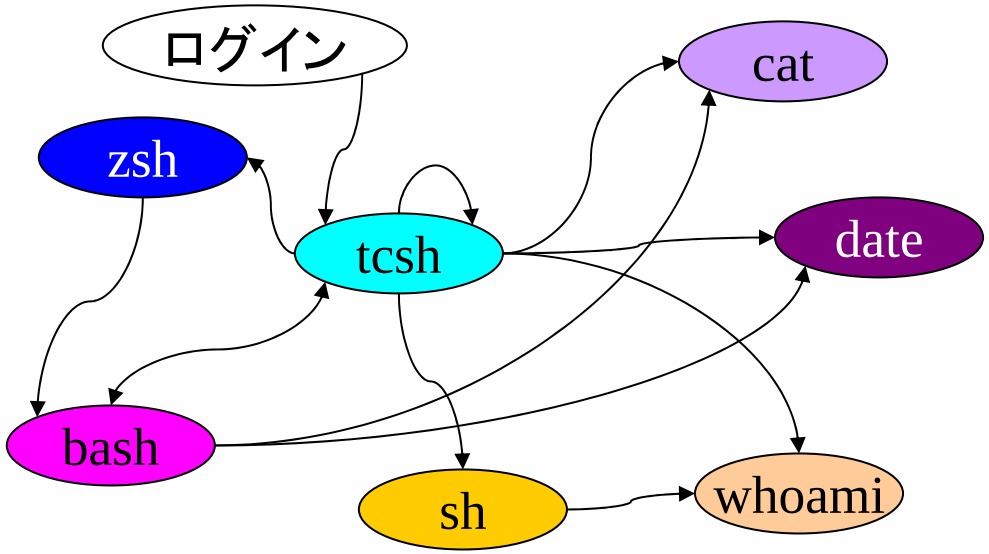
＋　実際のアクセスに基づく自動設定

⇒不要な許可を与える心配が無い

＜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）

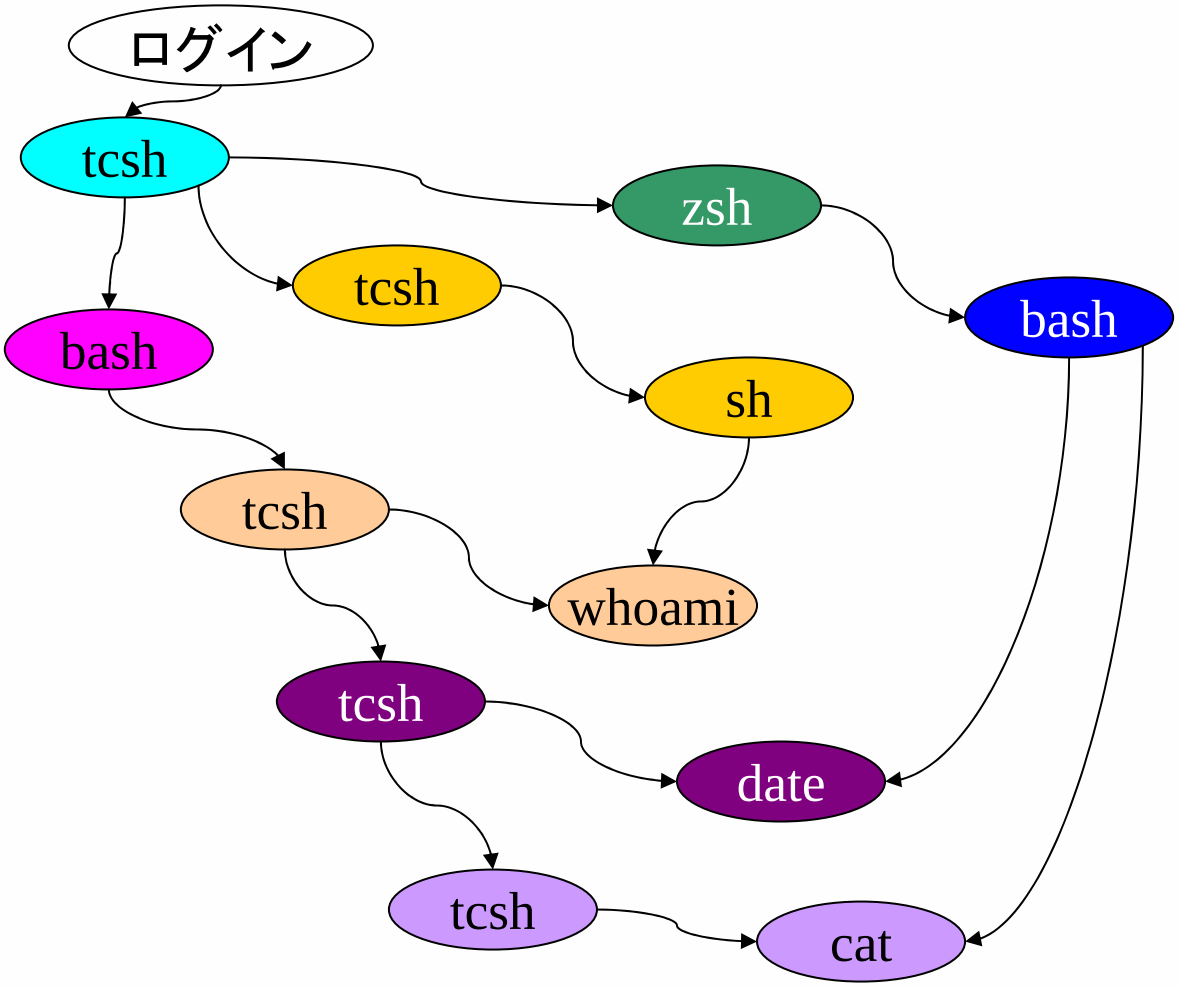


必要なドメインの数は少ないが、循環によって余計な許可を与えている。

- 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

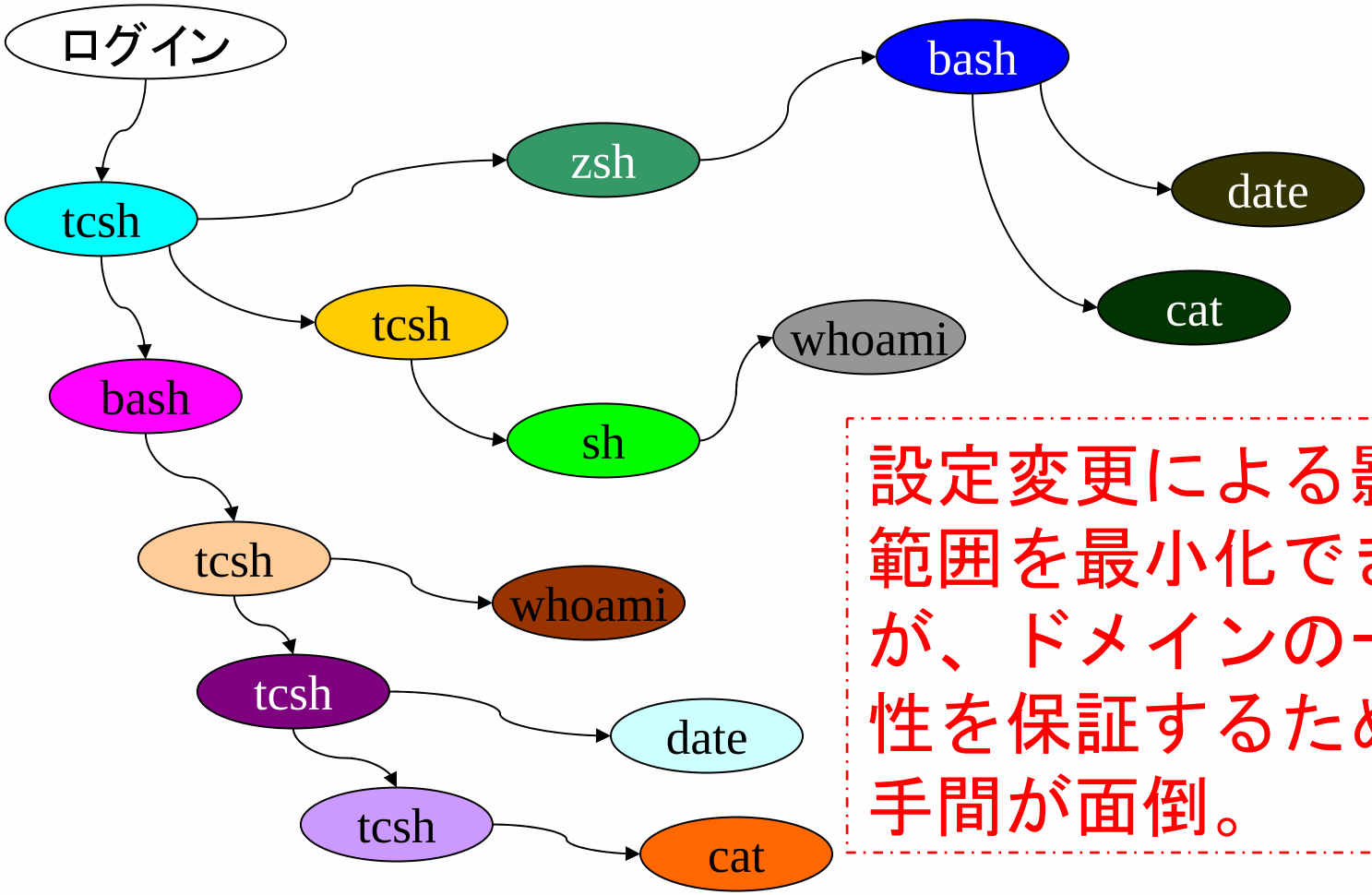
⇒最低限必要な条件は満たしている。

＜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）



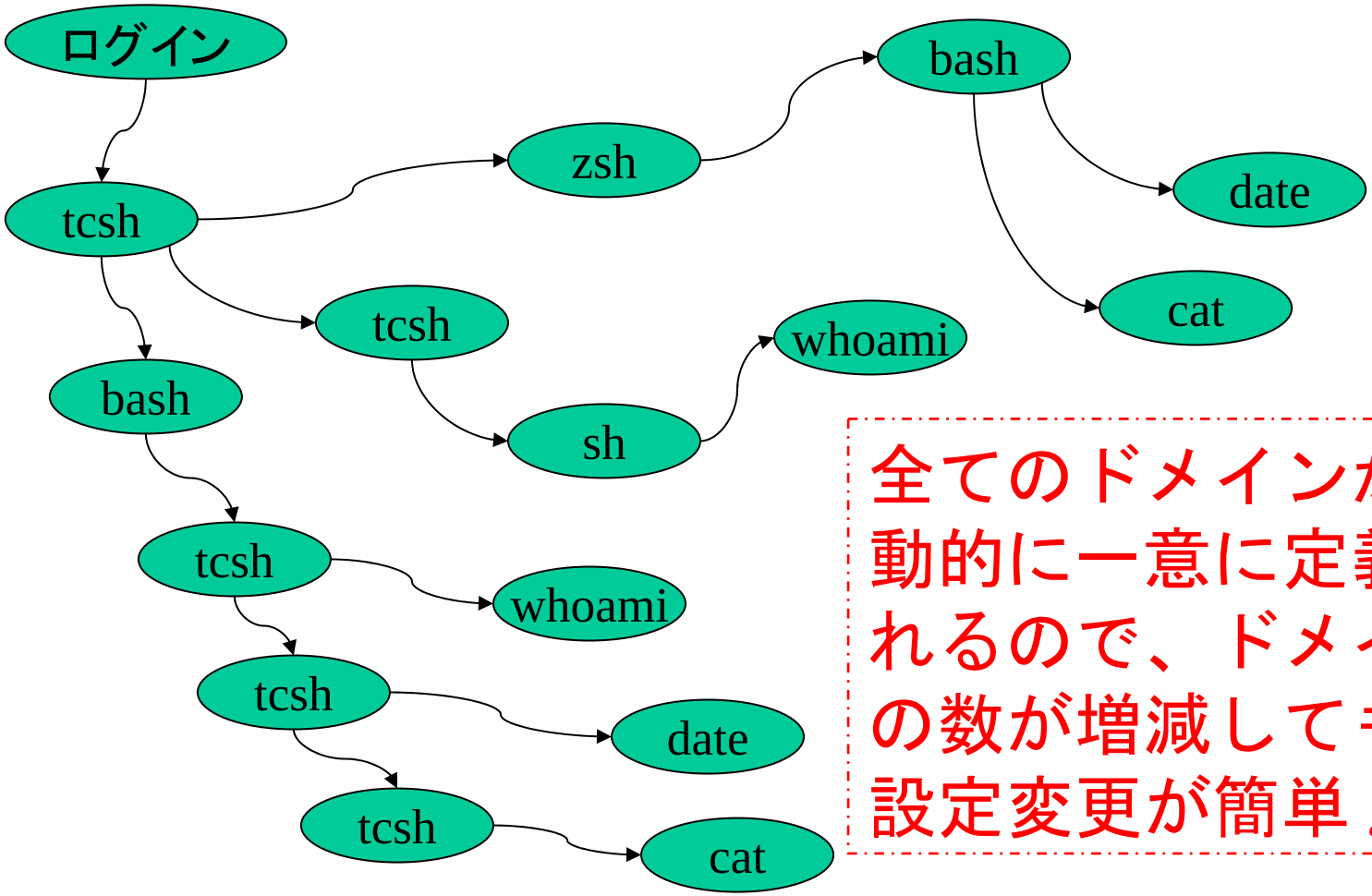
循環が生じないため余計な許可を与えず、必要なドメインの数も少ないが、設定変更による影響範囲に注意が必要。

＜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。

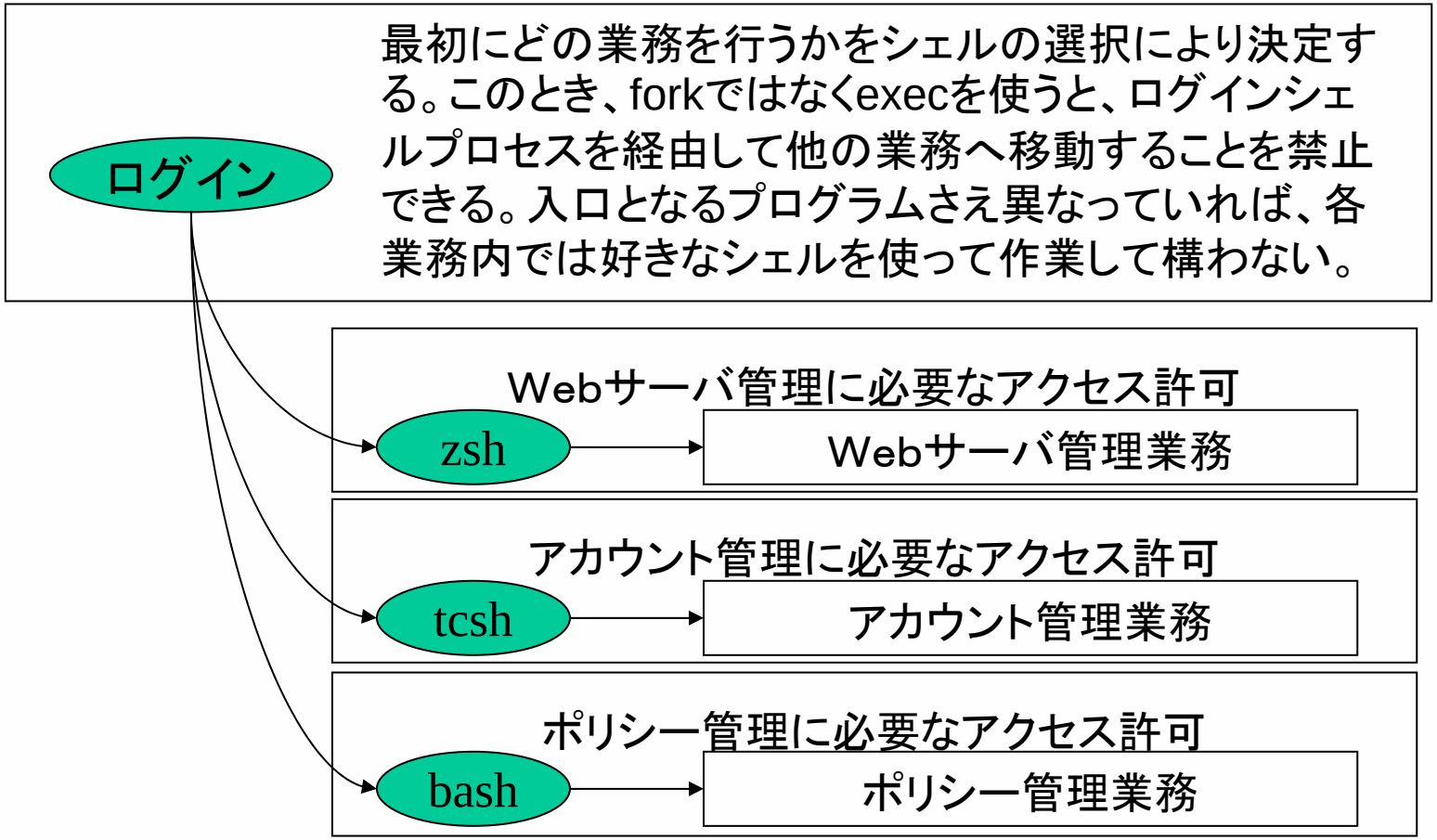
＜設定例4＞余計な許可を容認できない場合
（ログインを基点とするアクセス制御方式）



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♪

応用例1:ドメイン定義を利用したRBACもどき

業務毎に異なる入口プログラムを割り当て、その業務に必要なアクセス許可を与えることで、同一アカウントに対してRBACのような分割を行うことができる。



応用例2：セカンドパスワードもどき

ドメイン遷移のパターンを秘密にしておくことを前提に、重要な作業は複雑なドメイン遷移を経ないと実行できないようにすることで、パスワードクラッキング等により不正にログインされても簡単には重要な作業を実行させないようにできる。以下の例ではシェルを使用しているが、forkまたはexecにより他のプログラムを実行できるプログラムならば何でも構わない。