使いこなせて安全なLinuxを目指して
[当日説明用資料]

平成17年6月2日
株式会社NTTデータ
オープンソース開発センタ
技術開発担当
原田季栄
haradats@nttdata.co.jp

「安全なLinux」を目指す理由

- Linuxがもともと備える任意アクセス制御(DAC)だけでは安全ではなく、不十分だから。
- 具体的には？
  - バッファオーバフロー等による乗っ取りを受け、システム管理者権限を奪われると、歯止めがない。

「安全」ということ

- 何故「安全」が必要か？
  - 「高度情報化社会」という名のブラックボックス。
  - コンピュータシステムと無関係には生活できない。
  - エラーは完全に避けることができない。
- 高度な「理想」ではなく、最低限の「必要性」
  - システムが完全に乗っ取られない。
  - データが改ざんされない。
- Linuxは安全か？
  - 少なくとも標準のLinuxでは、NO
  - SELinuxでは原理的にはYes、実際には？

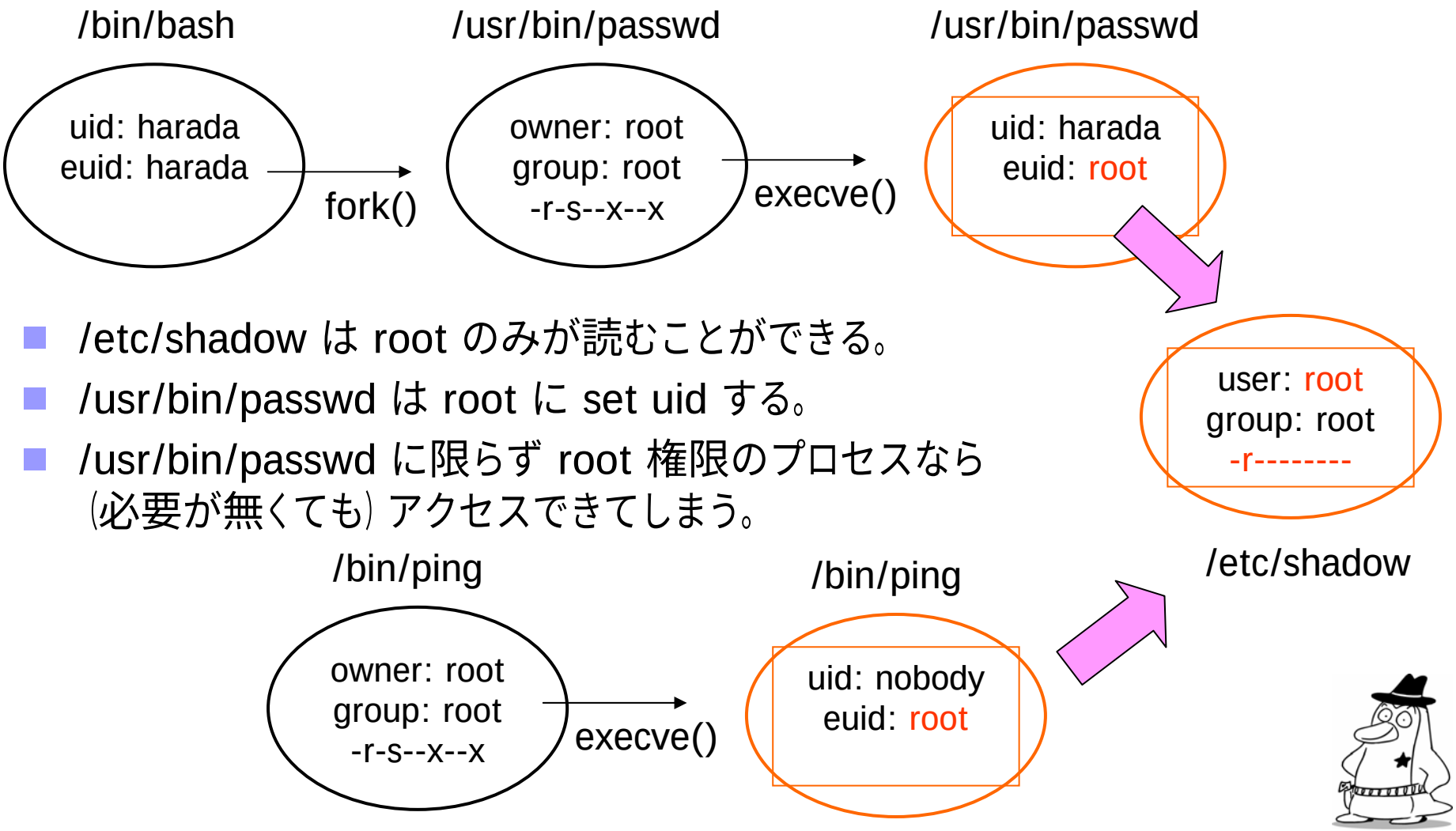
「使いこなせる」について

- 「使いこなせる」とは、どのようなことか？
  - 状態を自在に把握できる。
  - 思いのままに操り制御できる。
- 「使いこなせる」かどうかの指標は何か？
  - 概念のわかりやすさ
  - ポリシーの書きやすさ
  - 管理運用の手順

SELinuxという選択肢

- Linux 2.6カーネルには、SELinuxが組み込まれました。(2003.8)
  - 誰でも簡単に利用することができます。
  - Fedora Core 3, Red Hat Enterprise Linux 4.0 では、インストールするとデフォルトでSELinuxが有効となります。
- でも・・・・
  - 概念がとても複雑です。
  - 概念を理解できたとしても、実際にポリシーを管理するのが大変です。
  - 「自分はSELinuxを完全に使いこなせる」と断言できる人は、おそらく多くありません。
- 具体的な例を見てみましょう。

通常のLinuxの場合



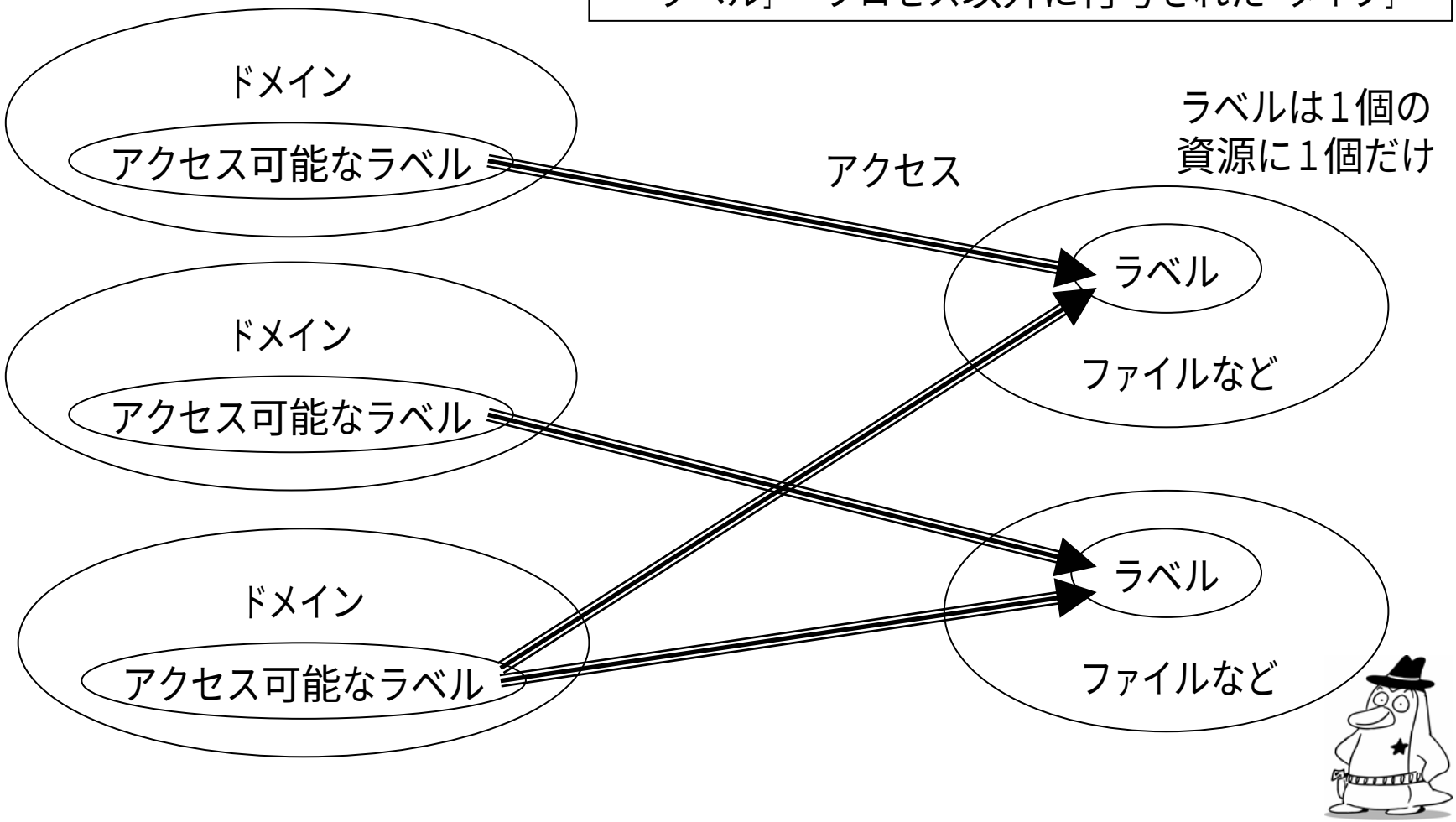
- /etc/shadow は root のみが読むことができる。
- /usr/bin/passwd は root に set uid する。
- /usr/bin/passwd に限らず root 権限のプロセスなら (必要が無くても) アクセスできてしまう。

SELinuxの場合 (TE)

プロセスはいずれかの
ドメインに所属する

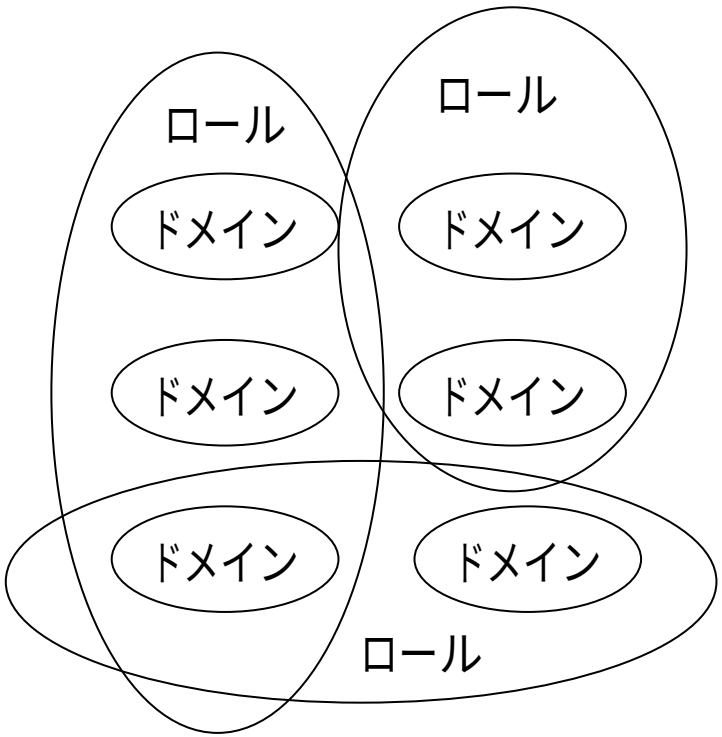
「ドメイン」＝プロセスに付与された「タイプ」

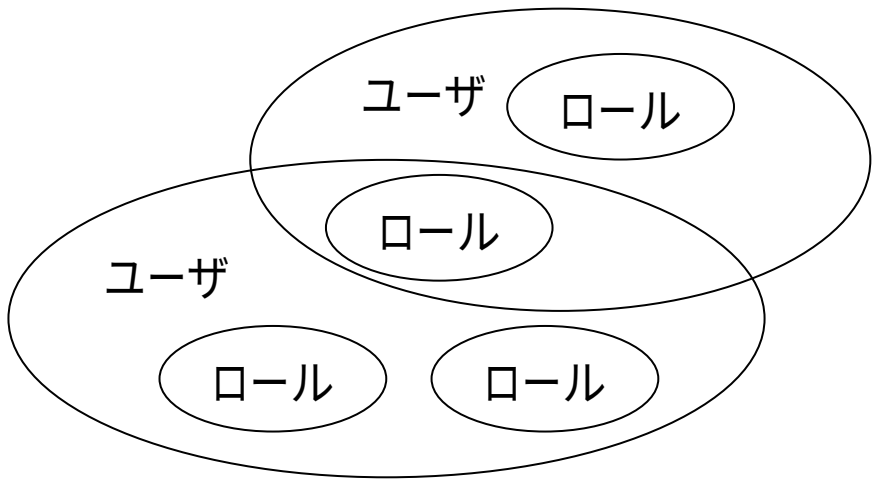
「ラベル」＝プロセス以外に付与された「タイプ」



SELinuxの場合 (RBAC)

SELinuxには、TEだけでなく、RBAC (Role-Based Access Control)も実装されています。





割り当てられたロールの範囲内で
複数のドメイン間を切り替えできる。

ユーザに割り当てられた範囲内で
複数のロール間を切り替えできる。

SELinuxの場合（概念）

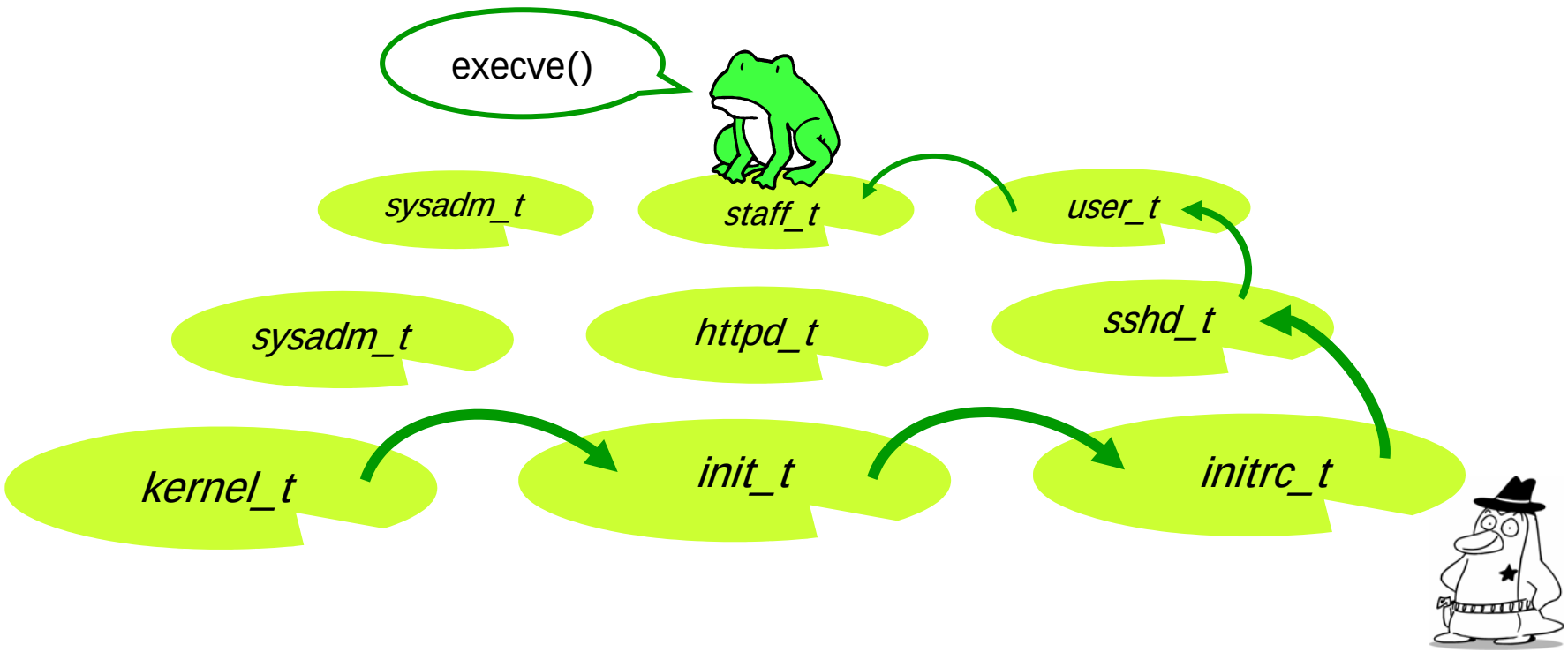
- 「ドメイン」と「ラベル」（TE：Type Enforcement）
  - 状況に応じたきめこまかなアクセス権限（Access Vector）を定義するために、資源に対して「タイプ」という名札が割り当てられる。「タイプ」のうち、プロセスにつけられたタイプを「ドメイン」、プロセス以外につけられたタイプを「ラベル」と呼ぶ。「ドメイン」はアクセスを制御したい範囲毎に定義するものであり、「アクセス許可定義の集合体」と考えることができる。プロセスは必ずドメインに属している。
- 「ロール」（RBAC：Role-Based Access Control）
  - 用途や目的毎の権限分割を行うために、複数の「ドメイン」を組み合わせて「ロール」（役割）というグループを構成することができる。SELinuxが導入されたシステムでは、「ユーザ」には必ず１つ以上の「ロール」が割り当てられる。「ユーザ」は自分に割り当てられている「ロール」群の中から、用途や目的に応じて適切なロールを選択して作業を行うことができる。
  - 「ユーザ」、「ユーザ」が属する「ロール」、「タイプ」の３要素を組み合わせた「セキュリティコンテキスト」と呼ばれる概念を導入しており、 RBACを詳細に設定できる。

SELinuxにおけるドメインの考え方

ドメインは階層構造を持たずにフラット。

プログラムの実行をトリガーとして他のドメインに遷移する。

ドメインは、ユーザ定義。



SELinuxの場合（構文）

「主体」となるプロセスのタイプ

「客体」となるオブジェクトのタイプ

allow passwd_t shadow_t : file {read write ... };

「客体」となるオブジェクトの種類（オブジェクトクラス）

許可するアクセスの内容（オブジェクトクラスごとに指定できる内容が異なる）

タイプについて、あらかじめ正しく割り当てられていることが前提となります。（タイプの定義自体もポリシーの一部）

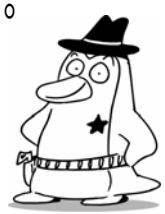
SELinuxの場合（ポリシー例）

- allow user_t passwd_exec_t : file { getattr execute };
  - /usr/bin/passwd コマンドを実行できるようにする。
  - /usr/bin/passwd に passwd_exec_t という型（タイプ）が付与されていることが前提。
- allow passwd_t passwd_exec_t : file entrypoint;
- allow user_t passwd_t : process transition;
  - /usr/bin/passwd の実行により user_t ドメインにあるプロセスを passwd_t ドメインに遷移させる。
- allow passwd_t shadow_t : file { read write append ... };
  - /usr/bin/passwd に /etc/shadow に対する read 等を認める。
  - /etc/shadow に shadow_t という型が付与されていることが前提。

SELinuxは使いこなせるか？

- 「使う」の意味によります。
- 「ポリシーのエラーが出ない」ようにするだけなら簡単です。
  - “permissive”（エラーがあっても処理を継続）モードで動かす。
  - ポリシー違反エラーをポリシー形式に変換する。（そのためのツールがありますが、RSA 2005で講演されたFrank Mayerさん曰く、 ”Don’t use it!”）
  - 変換したポリシーを反映する。
  - “enforcing”（ポリシーにないものは実行しない）モードで動かす。

    上記の処理を繰り返すと、考えることなく「エラーなしでSELinuxが動いている状態」が実現できます。しかし、
  - そのようにして使っていると
    - ポリシーのエラーがなくても改ざん等の被害を受ける可能性があります。
    - 被害を受けた場合に原因の特定が困難です。



SELinux運用の理想と現実

- 理想的には
  - システムの挙動をくまなく把握した上で、システム全体のポリシーを定義する。
  - ポリシーエラーが発生した場合、その内容を理解した上で、ポリシー(全体)を再定義する。
- 理想を阻む要因
  - ポリシーのファイルは何万行にも及びます。
  - ファイル名やディレクトリ名ではなく、それらに付与されたラベルに基づいて定義しなければなりません。
  - ポリシーの記述レベルは、システムコールと同等です。
- 運用の実態
  - 実装と一緒に配布されているデフォルトポリシーがベース。
  - 定義済みドメインそのままか、わずかな修正で使う。

SELinuxのまとめ

- 「使いこなせれば安全にできる」のですが、「使いこなすのが大変」というのが実情です。
- MITRE、Tresys、日立ソフトウェアエンジニアリング等により各種のツールが開発され、またSELinux自体進化していくので、この問題は徐々に改善されることを期待しています。
  - “New Reference Policy” Project（Mayer氏RSA講演より）
    - ポリシーをモジュールとして扱える。
    - 定義内容の意味がわかるようなマクロ言語を開発する。
  - NTTデータでも使い勝手を改善するためのツールを開発中です。
- でも、「今すぐ使いこなせて安全なLinuxを実現できないかな」と思い、取り組んできた結果が、TOMOYO Linuxです。

TOMOYO Linuxへの道のり

TOMOYO Linuxへの道のり(1)

- 目標設定：「物理的な改ざん防止」
  - 「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>
    - 物理的な改ざん防止＋マウント制限
    - 「ポリシーの運用なしに堅固に守ることができる」はずだったが・・・、
  - セキュリティ・スタジアム2004に「防御側」として出展
    - http://www.jnsa.org/seminar_20041101.html
    - Samba脆弱性を突いて管理者権限を奪われてしまいました。(想定内)
    - 「改ざん」はされませんでしたが、書き込み可能なパーティションにJavaプログラムを置かれて、偽のWebサーバを立てられてしまいました・・・。
    - /dev配下のファイルを削除されて、ログインができなくなってしまいました。
  - セキュリティ・スタジアムからの教訓
    - 「アクセス制御の強化」は、「安全なLinux」を実現する上で必要不可欠。
    - /devは防御が必要。

TOMOYO Linuxへの道のり(2)

- 目標設定：「ポリシーの自動定義」
  - 「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>
    - Linuxにおけるプロセス管理の機構に注目し、/sbin/init に始まるプロセス起動履歴毎にアクセス要求を記憶する仕組みを実装。
    - SubDomain風のポリシーの変換に成功。
  - 「TOMOYO Linux-タスク構造体の拡張によるセキュリティ強化Linux」
    - Linux Conference 2004 <http://lc.linux.or.jp/lc2004/03.html>
    - 2.4カーネルをベースにポリシー自動学習機能を備えたTOMOYO Linuxを開発。

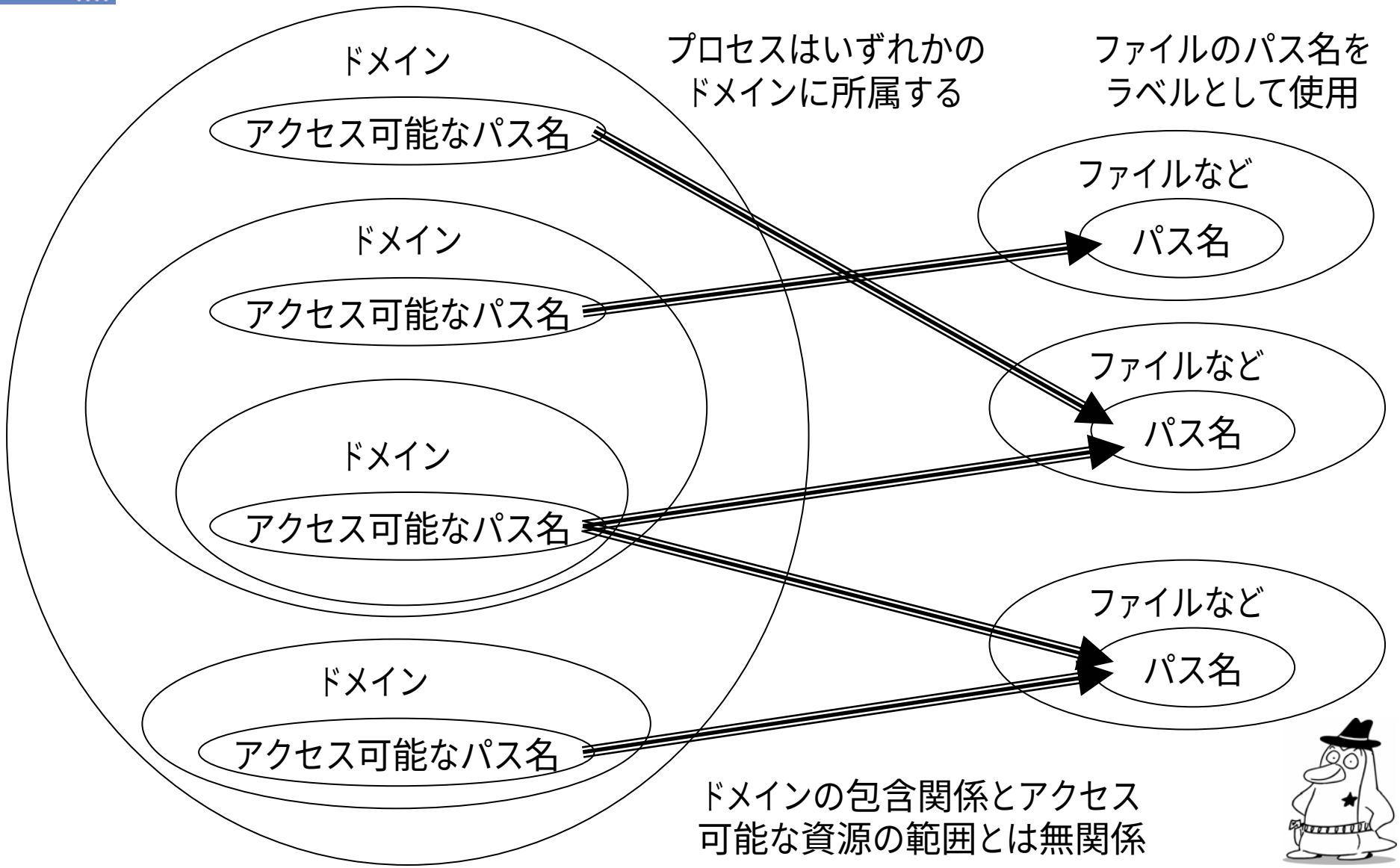
TOMOYO Linuxへの道のり(3)

- LC2004以後の開発
  - 「ファイルに対するアクセス許可」以外の機能追加
  - 物理的改ざん防止機能(SAKURA)の統合
  - 2.6カーネル対応
  - 独自のデバイスファイルシステム(SYAORAN)
  - ポリシー構成と仕様の見直し
    - 次ページ以降で説明します。

TOMOYO Linuxの概念

- プログラムの実行 (execve) 毎にドメインを自動的に定義する。
  - SELinuxではドメインの定義は、管理者裁量になります。
- ドメイン毎に、プロセスがアクセスする対象をファイル、ディレクトリ名そのままにread, write, executeの粒度でポリシーとして記述する。
  - SELinuxでは、30種のオブジェクトクラス毎に数十の粒度で権限を記述します。
- 「ポリシーの学習モード」と「ポリシーに基づくアクセス制御モード」を備える。
  - SELinuxでは、ポリシーの学習は対応していません。

TOMOYO Linuxの場合



ドメインの包含関係とアクセス
可能な資源の範囲とは無関係

TOMOYO Linuxにおけるドメイン

<kernel> (**カーネルのドメイン**)

<kernel> /sbin/init (**カーネルから起動されたinitのドメイン**)

<kernel> /sbin/init /etc/rc.d/rc

(**カーネルから起動されたinitから起動されたrcのドメイン**)

<kernel> /sbin/init /etc/rc.d/rc /etc/rc.d/init.d/httpd

(**カーネルから起動されたinitから起動されたrcから起動されたhttpdのドメイン**)

<kernel> /sbin/init /etc/rc.d/rc /etc/rc.d/init.d/httpd /sbin/initlog

(**カーネルから起動されたinitから起動されたrcから起動されたhttpdから起動されたinitlogのドメイン**)

<kernel> /sbin/init /etc/rc.d/rc /etc/rc.d/init.d/sshd

(**カーネルから起動されたinitから起動されたrcから起動されたsshdのドメイン**)

<kernel> /sbin/init /etc/rc.d/rc /etc/rc.d/init.d/sshd /sbin/initlog

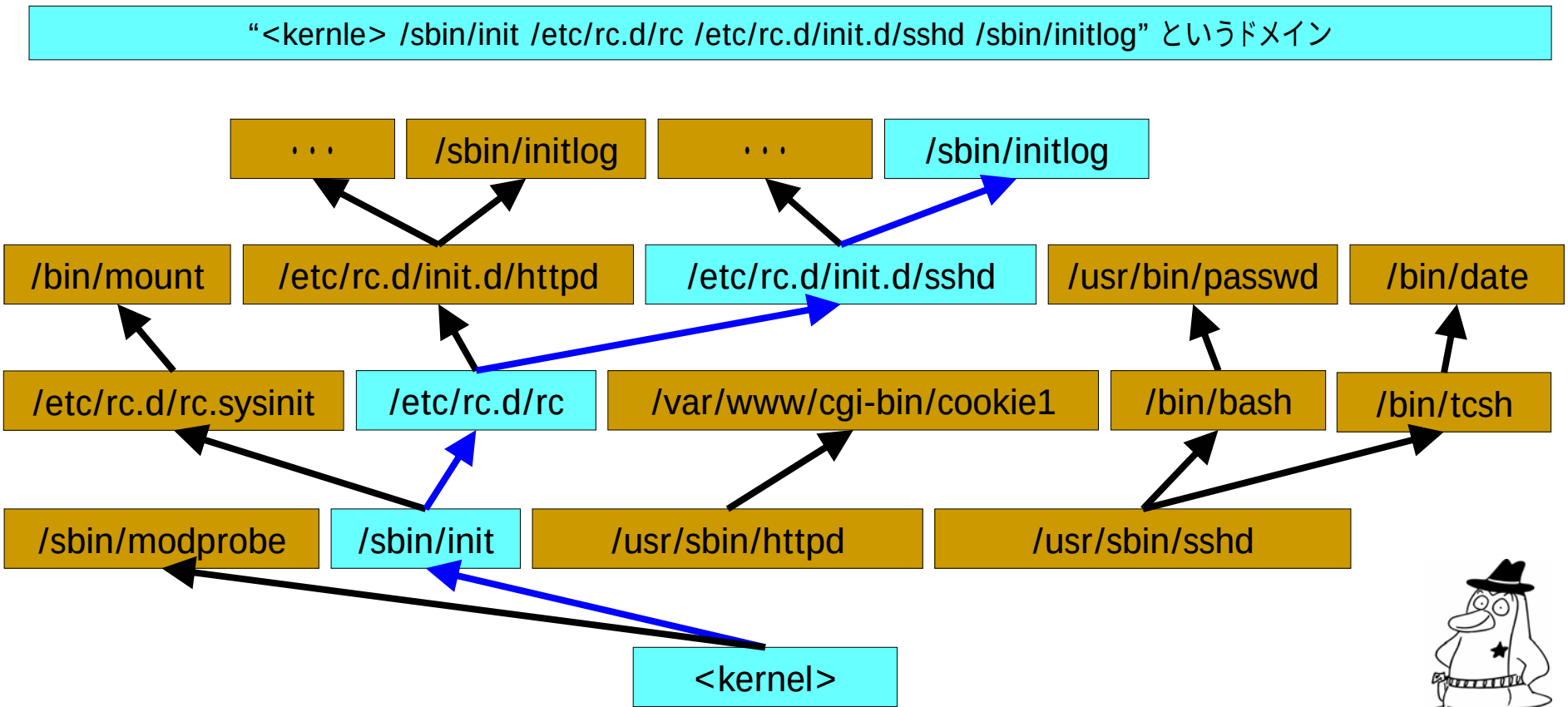
(**カーネルから起動されたinitから起動されたrcから起動されたsshdから起動されたinitlogのドメイン**)

プログラムの実行をトリガーとして自動的にドメインを定義して遷移。

TOMOYO Linuxにおけるドメイン

プロセスの実行履歴毎に独立なドメインを自動的に定義。

全てのプロセスを一意に特定することができる。



TOMOYO Linuxの場合（ポリシー例）

「主体」となるプロセスのドメイン→プロセス実行履歴毎

<kernel> /usr/sbin/sshd /bin/bash

1 /usr/bin/passwd

「客体」となる
オブジェクトのパス名

<kernel> /usr/sbin/sshd /bin/bash /usr/bin/passwd

6 /etc/shadow

認めるアクセス許可内容
読み込み：4
書き込み：2
実行：1

ドメインの包含関係より
このドメインに遷移できるのは
“<kernel> /usr/sbin/sshd /bin/bash” だけ。
他のドメインからこのドメインに
遷移してしまう心配は無用。

TOMOYO Linuxの場合（ポリシー例）

前ページのTOMOYO Linuxのポリシー

```
<kernel> /usr/sbin/sshd /bin/bash
    1      /usr/bin/passwd
<kernel> /usr/sbin/sshd /bin/bash /usr/bin/passwd
    6      /etc/shadow
```

を「SELinux風」に記述したとすると・・・（ファイルコンテキスト定義は除く）

```
type “<kernel> /usr/sbin/sshd /bin/bash”, domain;
type “<kernel> /usr/sbin/sshd /bin/bash /usr/bin/passwd”, domain;
type “/usr/bin/passwd”, file_type, exec_type;
type “/etc/shadow”, file_type;
domain_auto_trans(
    “<kernel> /usr/sbin/sshd /bin/bash”, “/usr/bin/passwd”,
    “<kernel> /usr/sbin/sshd /bin/bash /usr/bin/passwd”);
neverallow ~“<kernel> /usr/sbin/sshd /bin/bash”
            “<kernel> /usr/sbin/sshd /bin/bash /usr/bin/passwd”:process transition;
neverallow “<kernel> /usr/sbin/sshd /bin/bash /usr/bin/passwd”
            ~“/usr/bin/passwd”:file entrypoint;
allow “<kernel> /usr/sbin/sshd /bin/bash /usr/bin/passwd”
        “/etc/shadow”:file { read write };
```

TOMOYO Linuxのポリシー構成

ポリシー定義ファイル種別	ファイル名	作成単位	説明
ファイルに対するアクセス許可	policy.txt	ドメイン毎	次ページで例を紹介
bindを認めるポート	allow_bind.txt		
ケイパビリティ	cap_policy.txt		ネットワークを含む
「信頼済み」ドメイン	authorized.txt	システム全体	
「ドメイン遷移例外」	initializer.txt		
全てのドメインから読めるファイル一覧	allow_read.txt		共有ライブラリ等
chrootによる移動を認めるディレクトリ	chroot.txt		改ざん防止機能用
マウント許可情報	mount.txt		
アンマウント許可情報	noumount.txt		
作成を許可するデバイスファイルに関する情報	syaoran.conf		

※それぞれのポリシーファイルの意味と詳細については、論文を参照ください。

TOMOYO Linuxのポリシー記述例

ポリシー本体

```
<kernel>
1 /sbin/init
<kernel> /sbin/init
6 /dev/console
6 /dev/initctl
6 /dev/tty¥$
4 /etc/inittab
6 /etc/ioctl.save
4 /etc/localtime
1 /etc/rc.d/rc
1 /etc/rc.d/rc.sysinit
1 /sbin/mingetty
1 /sbin/shutdown
2 /var/log/wtmp
```

意　味

<kernel> は以下のことができる。
- /sbin/init を実行 (--x)

<kernel> から起動された /sbin/init は以下のことができる。
- /dev/console の読み書き (rw-)
- /dev/initctl の読み書き (rw-)
- /dev/tty[0-9]* の読み書き (rw-)
- /etc/inittab の読み込み (r--)
- /etc/ioctl.save の読み書き (rw-)
- /etc/localtime の読み込み (r--)
- /etc/rc.d/rc を実行 (--x)
- /etc/rc.d/rc.sysinit を実行 (--x)
- /sbin/mingetty を実行 (--x)
- /sbin/shutdown を実行 (--x)
- /var/log/wtmp の書き込み (-w-)
- /var/run/utmp の読み書き (rw-)

「信頼済みドメイン」について

- 強制アクセス制御が有効な状態にあっても、Linux標準の任意アクセス制御しか受けないドメイン。
- 想定する利用形態
  - ソフトウェアのアップデート（rpmの実行）等、どのようなアクセス許可が必要かを事前に知ることができない場合。
  - デーモンプロセスだけを強制アクセス制御で保護したい場合。（SELinux の Targeted モードに相当）
- 「強制アクセス制御を無効にせずにメンテナンスを行いたい」という導入現場のニーズから生まれた機能です。

「ドメイン遷移例外」について

- TOMOYO Linuxでは
  - プロセスの起動履歴に基づきドメインを自動的に定義します。
  - そのため、同じプログラムであっても、スクリプトから実行された場合とコマンドラインから実行した場合とでは異なるドメインとして扱われます。
  - これはTOMOYO Linuxの特長のひとつですが、サーバプログラムのように、複数の方法で起動されるものを同一のドメインとして扱いたい場合には不便です。
  - そこで、プロセスの起動履歴をリセットする場合を例外としてポリシーで記述できるようにしました。
  - 「サーバプログラムをコマンドラインから再起動させた場合もスクリプトから実行された場合と同一のドメインで扱いたい」という導入現場のニーズから生まれた機能です。

デバイスファイルの保護について

- 通常のLinuxの場合
  - CAP_MKNOD権限（管理者権限）を持つプロセスは任意のデバイスファイルを作成可能。
  - よって、管理者権限を奪われたら悪意あるデバイスファイルも作成可（例：/dev/sdaの属性を持った /dev/null という名前のデバイスファイル）
- SELinuxの場合
  - CAP_MKNOD権限を制限することでリスクを軽減。
  - しかし、CAP_MKNOD権限を持つプロセスならば悪意あるデバイスファイルも作成可能。
- TOMOYO Linuxの場合
  - CAP_MKNOD権限を制限するのではなく、/dev専用ファイルシステム(SYAORAN) で制限。
  - よって、管理者権限を奪われても悪意あるデバイスファイルを作成不可能。
  - /dev以外のパーティションではデバイスファイルのオープンを禁止可能。

「応用」と「デモ」

- 「使いこなせる」を目指して開発されたシンプルなTOMOYO Linuxですが、ここまで紹介した機能を用いて面白い応用が可能です。デモを交えてご紹介します。

TOMOYO Linuxの「応用」について

- TOMOYO Linuxは、SELinuxのような粒度の高いアクセス制御は行えません。
- しかし、工夫次第で面白い使い方が可能です。
  - なりすまし対策
  - RBAC (Role-Based Access Control)

応用1：TOMOYO Linuxによるなりすまし対策

- TOMOYO Linuxは、SELinuxのようにsshdを改造して「シェルコード対策」をすることはできません。
- しかし、簡単な工夫で「シェルコード対策」だけでなく「パスワード破り対策」もできます。
- 例えば、
  - sshでログイン後、特定のプログラムだけを実行できるようにする。
  - そのプログラムの中で追加の認証を行い、認証に成功した場合だけその他のプログラムも実行できるようにする。
  - 認証プロトコルはユーザが自由に定義できる。

応用2：TOMOYO LinuxによるRBAC

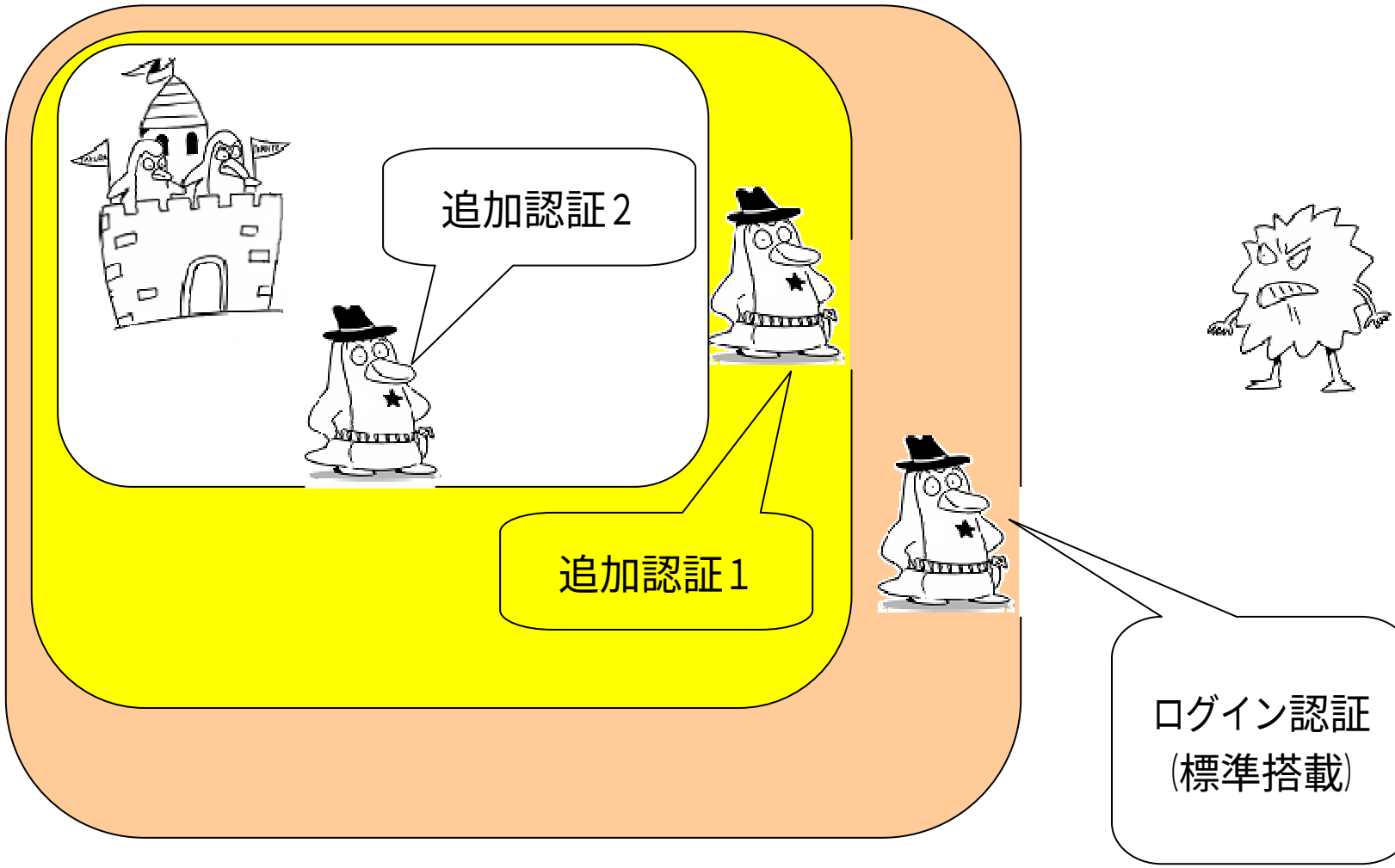
- TOMOYO Linuxは、SELinuxとは異なり、機能としてのRBAC (Role-Based Access Control)は実装していませんが、簡単な工夫で同様のことができます。
- 例えば、
  - tcshを起動した場合は「スタッフ」
  - bashを起動した場合は「セキュリティ管理者」
- ドメインを単位に自動的にグループ化されるので、ロールについて意識する必要がありません。

デモ：なりすまし対策

- ssh経由でrootとしてログインする。
- ログイン後にシェルが提供されるが、追加認証１を行う以外、何もできない。
- 追加認証１（別のパスワード＋追加要素を併用）をクリアし、次のシェルが提供されるが、追加認証２を行う以外、何もできない。
- 追加認証２（別のパスワード＋別の追加要素を併用）をクリアし、好みのシェルを起動して作業を開始。

- rootパスワードが漏洩しても全く問題にならないので、最初のログイン認証を突破されてから変更すればよい。
- 不正侵入者には諦めてログアウトしてもらう。

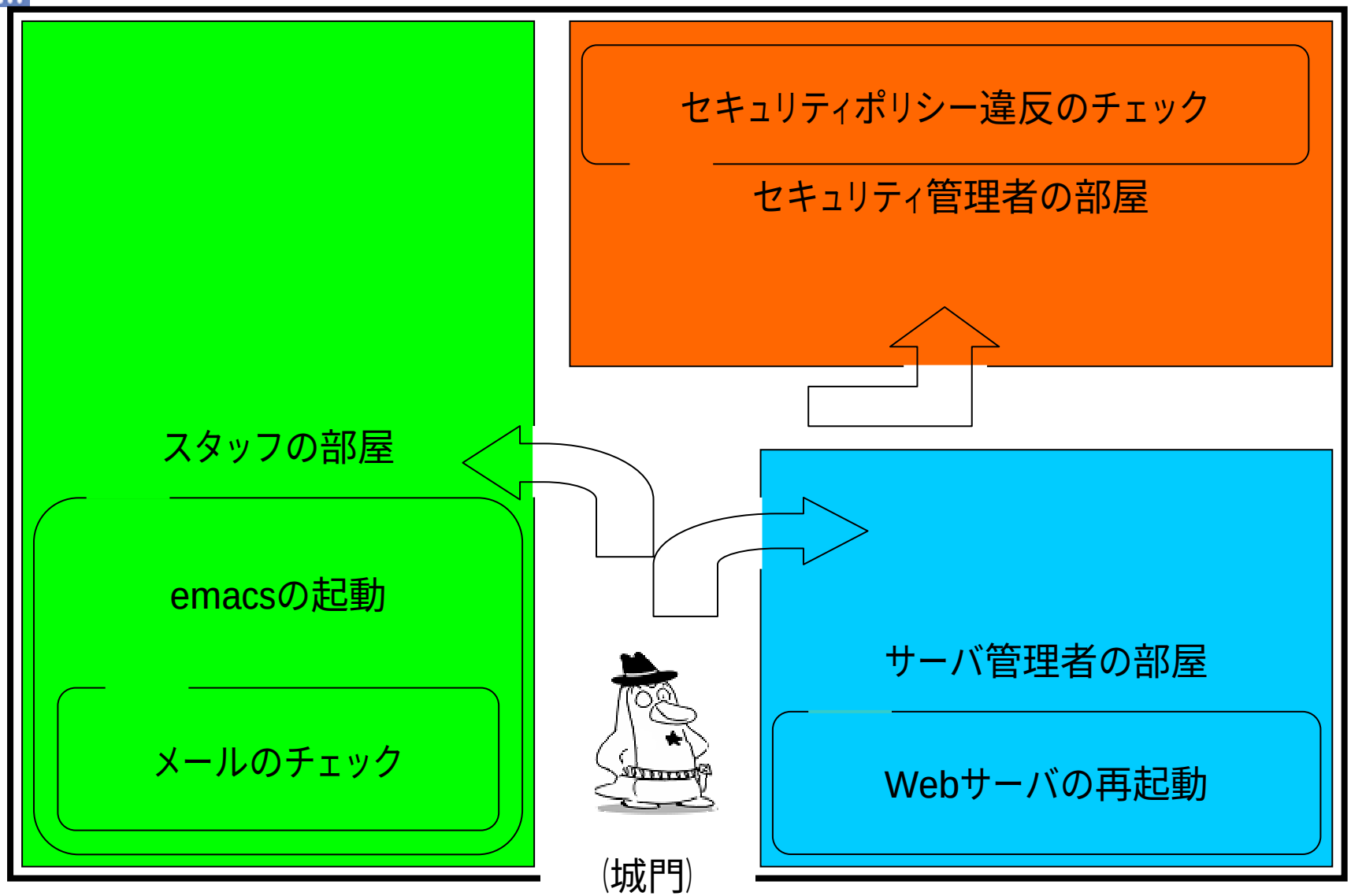
デモ：なりすまし対策



デモ：RBAC

- rootとしてログインする。
- zshを起動してサーバ管理者になり、Webサーバの再起動を行った後、zshから抜ける。
- tcshを起動してスタッフになり、emacsを起動してメールをチェックした後、tcshを抜ける。
- bashを起動してセキュリティ管理者になり、セキュリティポリシー違反が無いかをチェックした後、bashから抜ける。
- ログアウトする。

デモ：RBAC



開発者が考えるTOMOYO Linux

- Linuxセキュリティ強化の選択肢のひとつとして。
  - プロフェッショナル向けのSELinux
  - SELinuxほどのきめこまかな制御はできないが、誰でも簡単に使えるTOMOYO Linux
- 貢献として。
  - 実装
  - 考え方
  - 機能仕様 (特にポリシー)

おわりに

- 本説明資料の作成にあたっては、2005年5月11日に、日本SELinuxユーザ会とセキュアOS研究会主催により開催された「Frank Mayer氏と語る会」の講演内容を参考とさせていただきました。素晴らしい講演を行っていただいたMayer氏と会を企画してくれた方々に心より感謝致します。
- SELinuxの概念の整理について、辛抱強くコメントに対応いただきました碓井啓弘氏、平坂透氏に感謝します。
- 素晴らしい講演用のプレゼンテーション素材を作成いただいたアシュビーの中間ひとみ様、本資料に掲載しているTOMOYO Linuxのキャラクターを作成いただいた五十嵐晃様に感謝します。
- 構想検討の段階から、TOMOYO Linuxに関わる活動を変わらぬ情熱を持って支援いただいている半田哲夫氏に感謝します。

- 論文、本説明資料およびTOMOYO Linuxに関するご質問は下記メールアドレスにお送りください。
  - tomoyo-support@kits.nttdata.co.jp