Security-Enhanced Linuxの概要とxSPへの適用に向けて

2004.1.29

渡瀬 順平
NTT情報流通プラットフォーム研究所

本日の目標

- 情報共有
  - SELinux, セキュアOSって何？
  - できること, できないこと
- ディスカッション
  - どのように活用すれば効果的か？
  - 活用するために必要なこと
  - その他, インターネットのセキュリティに関して. . .

内容

- 現状認識とOSセキュリティの必要性
- SELinuxの概要
- 機能と構造
- 利点と限界
- パフォーマンスペナルティ
- xSPにおける使い方
- ディスカッション

現状認識とOSセキュリティの必要性

現状認識 (一般論)

- 情報セキュリティの重要性の高まり
- 従来型のセキュリティ対策の限界
  - Perimeter Protection (境界防御)
  - パッチマネージメント
- 完璧なセキュリティはない

情報セキュリティの重要性の高まり

- 不正行為の影響（不正者の利益）が増大
  - 重要な情報，サービスがネットワークに繋がる
    - オンラインバンキング，電子商取引
    - 電子政府（電子申請，住民基本台帳）
    - IP電話，システム監視・制御（交通，電力等）
- 不正行為をはたらくためのコストが低下
  - セキュリティホールの増加
    - アプリケーションの多様化
    - 品質，セキュリティよりもコスト，納期を優先する傾向
  - 管理が不十分なコンピュータの増加
    - スキルと時間を十分に持たない管理者が増加
    - “サーバ並”のホームPCがブロードバンドで常時接続
  - 不正アクセスに関する情報・ツールの入手性が向上
    - 技術のオープン化，インターネットによる情報流通

従来型セキュリティ対策の限界

- Perimeter Protectionの限界
  - 内部ネットワークが信頼できることが前提
  - Firewallで許可されているサービスは保護しない
    - 多くのアプリケーションは「TCPの80番」を使用できる
    - Firewallを通り抜けるトンネル技術が手軽に利用可能
      - L4のセッションイニシエーションは内→外
      - L7のセッションイニシエーションは外→内
  - Firewallの性能と検査精度はトレードオフ
    - ブロードバンド化により境界での精査が困難に. . .
  - モバイル，ワイヤレス，VPN，IPv6
    - エンドノードが境界に. . .

従来型セキュリティ対策モデルの限界

- パッチマネージメントの問題
  - 不定期に多大な検証稼動が必要
    - 現在は，オペレータの誠意と気合で何とかなっている？
  - パッチが必要かどうか判断するのが難しい
    - 外注して開発したシステム
    - 例えば，OpenSSLの脆弱性に関する情報を入手しても，自分のシステムのどのプログラムで使っているのか分からない
    - プリンタ，情報家電などの非PCデバイス
  - すぐに適用できない場合もある
    - 保守契約の内容によっては適用が難しい
    - パッチを当てると動かなくなるアプリケーション
    - ナローバンドユーザ
    - プリンタ，情報家電などの非PCデバイス
  - パッチ適用前の攻撃には対応できない

完璧なセキュリティはない

- 情報セキュリティ総合戦略（経産省,2002/10/10）
  - しなやかな「事故前提社会システム」の構築
    - 事故の回避（予防）
    - 被害の最小化・局限化
    - 回復力の確保
- 技術，マネジメントの両面から様々な取組みの積み重ねが必要
  - 交通システム等と同様の安全管理の仕組みが情報システム，ネットワークにも必要
    - マネジメント：法律，免許，車検，教育，保険，排出規制，etc
    - テクノロジー：バックミラー，ABS，シートベルト，エアバッグ，衝突安全ボディ，フライトレコーダ，etc

なぜ，OSのセキュリティ？

- エンドポイント防御が必要
  - ネットワーク境界だけでは守りきれない
  - アクセス制御負荷の分散
  - 守りたいものへのアクセスを直接，細かく制御
- 被害を最小化･局限化する機構が必要
  - 脆弱性のあるアプリケーション，不届きなユーザ，ヒューマンエラーは無くならない
- 詳細なロギングが必要
  - 説明責任，事故対応
- 信頼に基づく権限の部分委譲の仕組みが必要
  - リモートアクセス，アウトソーシング，プログラム自動アップデート

SELinuxの概要

SELinuxとは？

- Linuxカーネルに「”粒度の細かい”強制”アクセス制御機能」を追加するモジュール
- 米国のNSA (National Security Agency) が中心になって開発
- ポリシー管理モデルとしてTEとRBACを実装
  - Type Enforcement (TE)
  - Role-Based Access Control (RBAC)
- ユーザプログラムに対してトランスペアレント
  - 通常のLinuxとバイナリレベルの互換性
- GPL

歴史と動向

- 1992年～：NSAとSCCによる共同研究
  - DTMach，DTOS
- 1998年頃：ユタ大の研究用OSへ移植
  - Flask Architecture
- 2000年頃：Flask ArchitectureをLinuxへ移植
- 2000年12月：一般公開（linux-2.2ベース）
- 2003年9月：linux-2.6にマージ
- 現在：
  - Linuxディストリビューションへの取り込みが活発化
    - Fedora Core 2（2004年4月），RHEL4（2005年1Q），Debian(Woody)，Gentoo，SuSE
  - TrustedBSDのMACフレームワークへも移植作業中

アクセス制御の粒度

- クラス (リソース種別)
  - 31種類

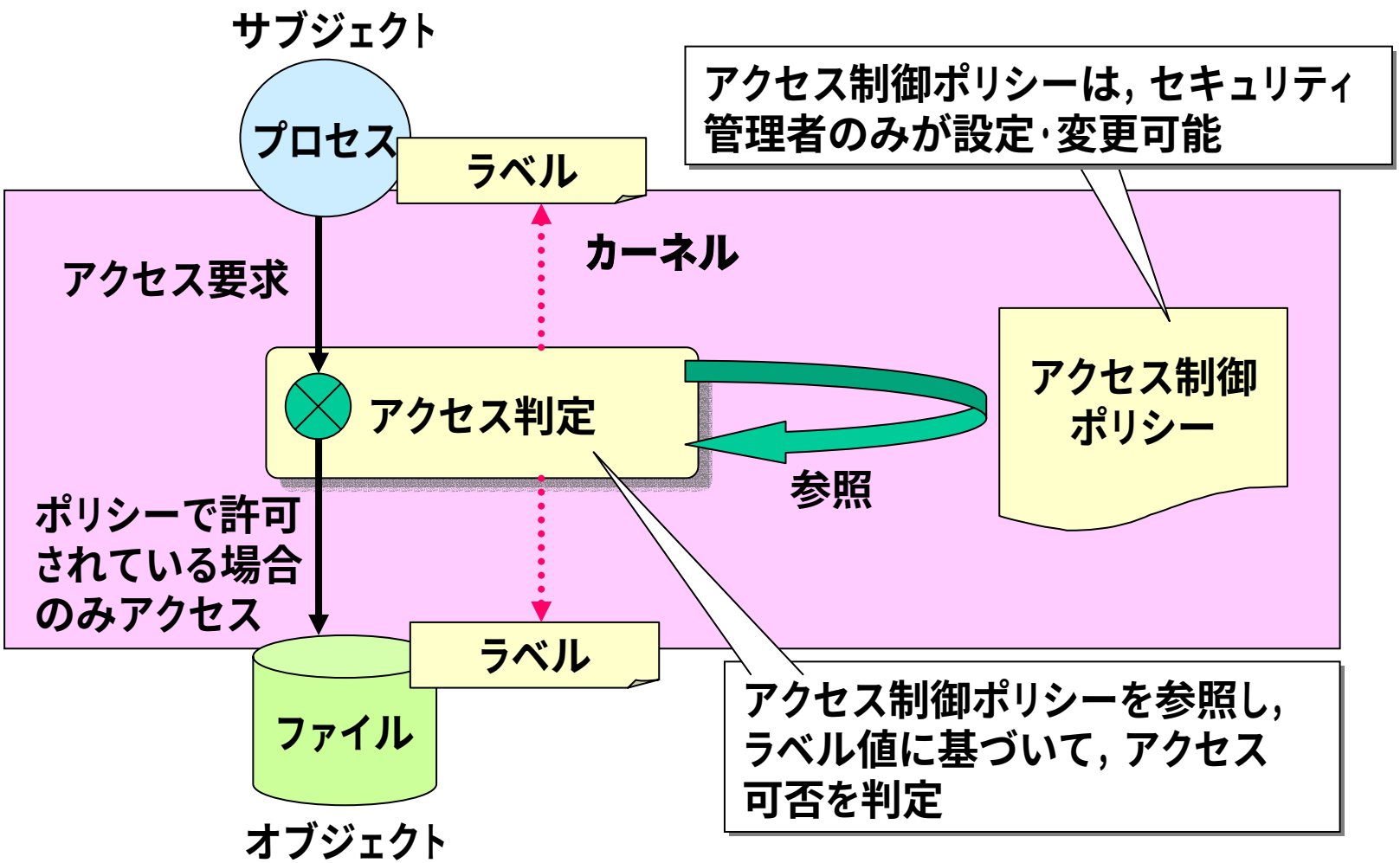
  process filesystem file dir fd lnk_file chr_file blk_file sock_file fifo_file socket tcp_socket udp_socket rawip_socket node netif netlink_socket packet_socket key_socket unix_stream_socket unix_dgram_socket sem msg msgq shm ipc passwd security system capability
- アクセスベクタ (パーミッション種別)
  - クラス毎に規定
  - fileクラスの場合は19種類

  ioctl read write create getattr setattr lock relabelfrom relabelto append unlink link rename execute execute_no_trans entrypoint quotaon swapon mounton

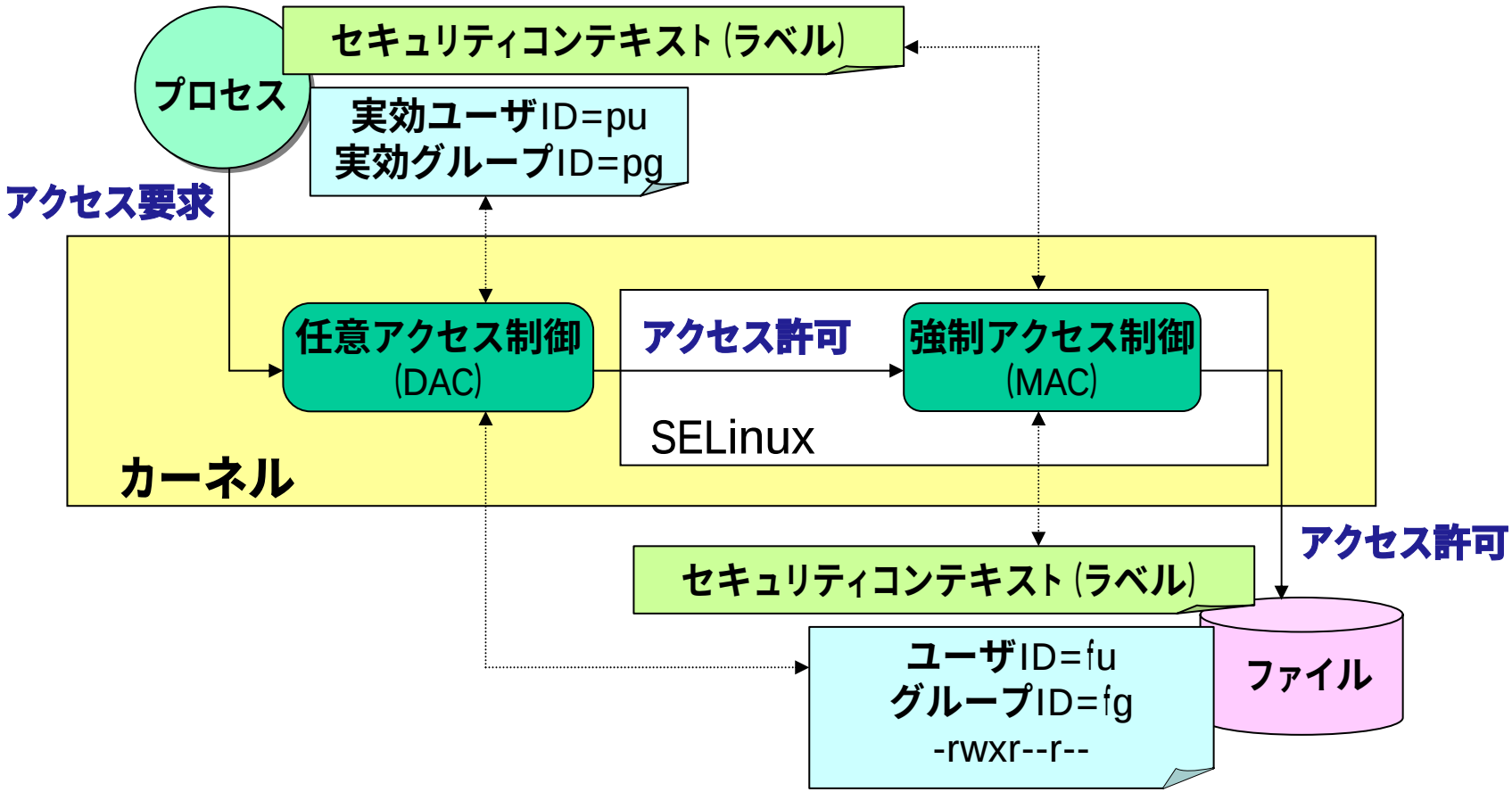
強制アクセス制御とは？

- 任意アクセス制御 (DAC)
  - 例えば，UNIXやWindowsの通常のアクセス制御
  - アクセス権をリソースのオーナが「任意に」設定可能
    - user，group，othersベースのパーミッション
    - POSIX ACL
  - UNIX，Windows，Linuxの場合，スーパユーザにはセキュリティチェックが働かないのが問題
- 強制アクセス制御 (MAC)
  - リソースのオーナではなく，セキュリティ管理者がアクセス制御を「強制」
  - スーパユーザも従わなければならない
  - rootの権限を分割して委譲することが可能

OSにおける強制アクセス制御の実装



SELinuxにおけるDACとMACの関係



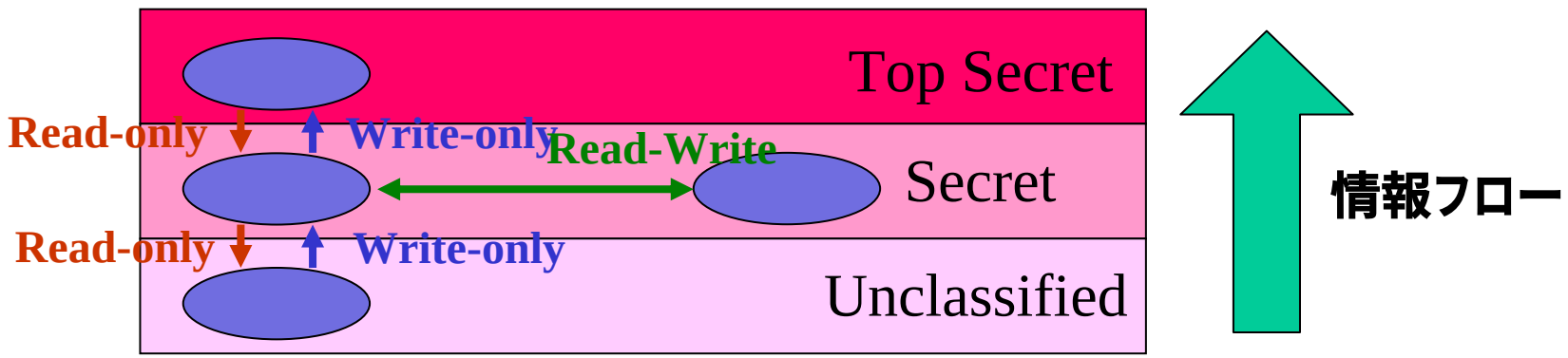
ユーザ，グループに基づく任意アクセス制御の後に，強制アクセス制御を実行．
両方のパーミッションが許可されている場合のみ，リソースへのアクセスが可能．

強制アクセス制御は新しい技術？

- 1980年代から使用されている枯れた技術
- 当初は，軍事用途向けの実装
  - TCSEC (Trusted Computing System Evaluation Criteria，1983,1985)
  - 階層的な組織における情報フロー制御とユーザ間・組織間の隔離を志向 (MLS，CMWS)
  - ポリシー (使い方) がメカニズムに侵食しており，汎用的な用途には向かなかった
- 一般的なオープンシステムやインターネットに適用しやすいようにアクセス制御モデルが発展
  - ポリシーとメカニズムの分離
  - 柔軟性の向上と引き換えにポリシーの検証が難しい

TCSECのアクセス制御モデル

- Multi Level Security
  - Bell-LaPadulaモデルによる情報フロー制御
  - 基本的な原理
    - NRU (Non-Read Up)
    - NWD (Non-Write Down)
  - 階層的な組織における情報フローを厳密に制御可能



汎用的に利用できるアクセス制御モデル

- 原始的なアクセス行列
  - 柔軟ではある
  - 粒度を細かくすると管理が煩雑
  - ポリシーの設定正常性の検証が難しい

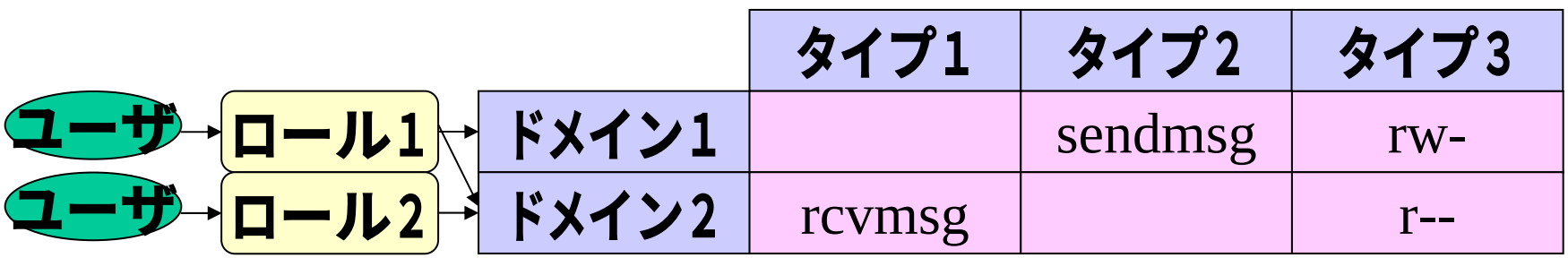
オブジェクト

	プロセス1	プロセス2	ファイル1	ファイル2
ユーザ1			rwx	rwx
ユーザ2			r-x	r-x
プロセス1		sendmsg	rw-	rw-
プロセス2	rcvmsg		r--	r--

サブジェクト

スケーラビリティと管理性を向上する工夫

- TE (Type Enforcement)
  - セキュリティの観点から等価なものをドメイン/タイプに集約し，マトリクスを縮小
  - ドメインからタイプへのアクセス制御ルールとしてセキュリティポリシーを表現
- RBAC (Role-Based Access Control)
  - 権限のセットとしてロールを定義
  - ユーザに対して，ロールを割り当て



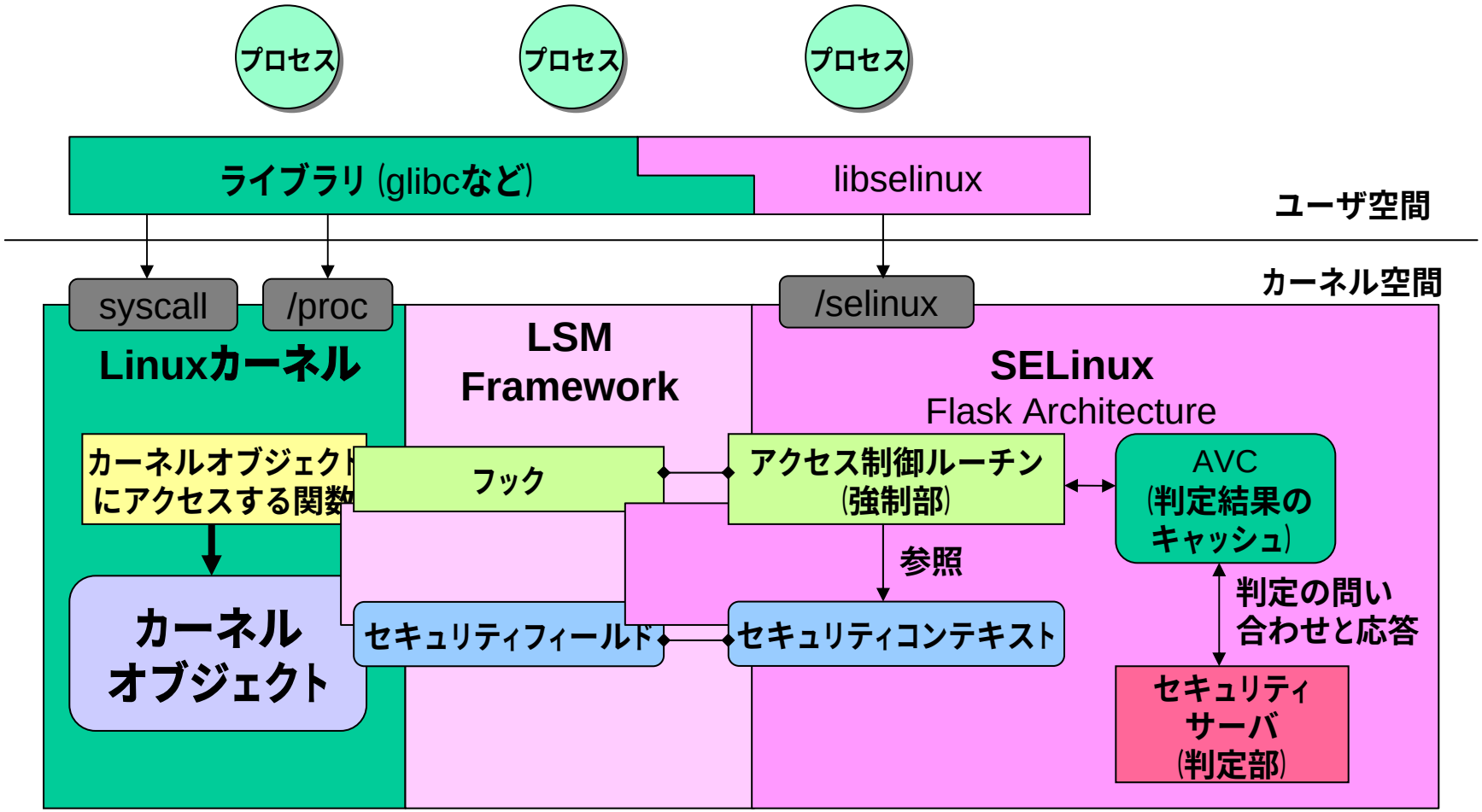
SELinuxの機能と構造

SELinuxの構成

- コンポーネント
  - カーネルモジュール
  - ライブラリ (libselinux)
  - ユーティリティ
  - 既存プログラムへのパッチ
  - サンプルポリシー

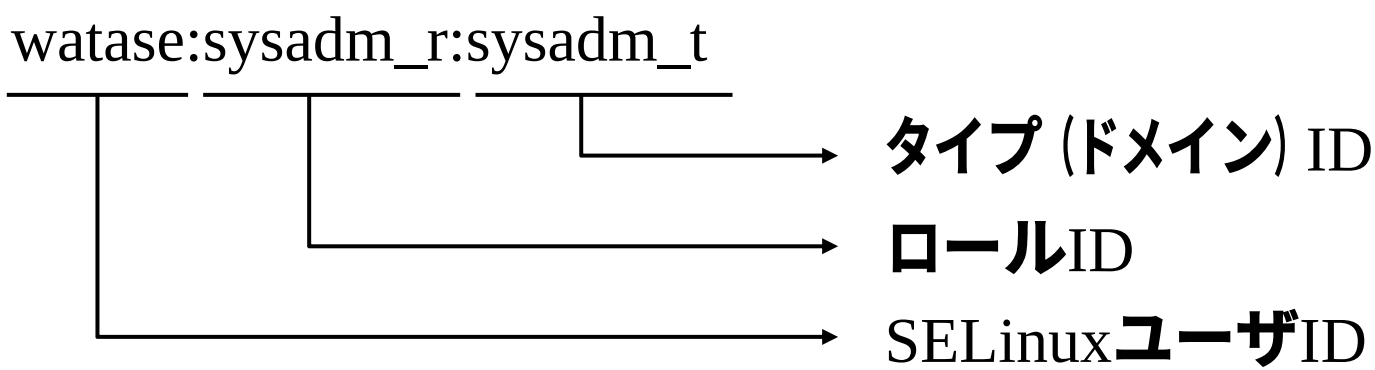
- LSM Frameworkを利用して実装
  - 様々なアクセス制御機能をLinuxカーネルに追加するためのフレームワーク

SELinuxの構造



LSM FrameworkとSELinuxは，Linuxカーネルのモジュールであり，SELinuxは，LSMFへのスタックとして実装されている．LSMFは，カーネルオブジェクトへのセキュリティフィールドの追加とアクセス関数へのフックを提供する．

セキュリティコンテキスト



- タイプ（ドメイン）ID
  - TEによるアクセス制御に使用
  - サブジェクト（プロセス）の場合はドメインと呼ぶ
- ロールID
  - RBACによるアクセス制御に使用
- SELinuxユーザID
  - LinuxのユーザIDとは独立に管理
  - suやSUIDプログラムを実行してもログインセッション中に変更されることはない

セキュリティコンテキストの管理

- カーネルオブジェクトを管理する構造体のメンバとして格納
  - プロセスディスクリプタのメンバ
  - ファイルディスクリプタのメンバ
  - 等々
- ファイルの永続的なラベルはディスク上のinode拡張属性に格納
  - 現在拡張属性が使用できるのはext2/ext3のみ
  - 他のファイルシステムには，パス名とラベルのマッピングDBを使用

SELinuxのType Enforcement

- ドメインからタイプへのアクセスを制御

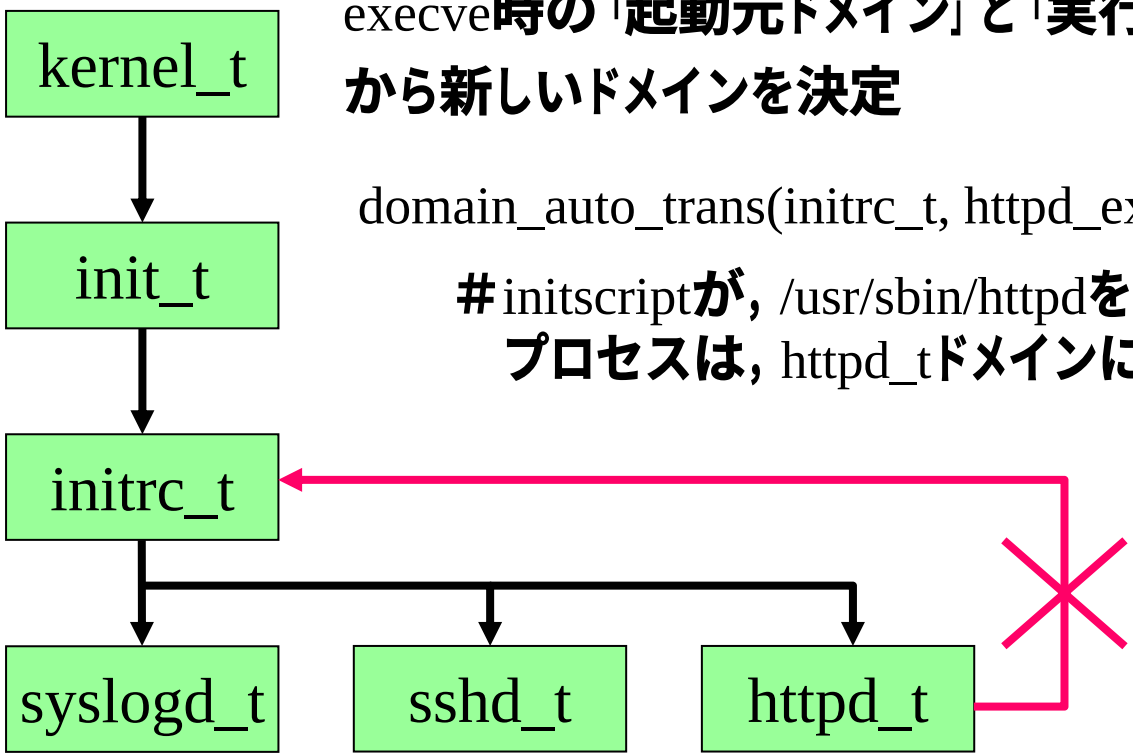
source type　サブジェクトのドメイン
target type　オブジェクトのタイプ
object class　オブジェクトのリソース種別
access vector　パーミッションのビットマップ表現

- 明示的に許可されているオペレーションのみ許可

allow httpd_t httpd_sys_content_t:file read

ドメイン遷移

- プロセスの動作するドメインを設定する仕組み
- 権限分割，プロセス間の隔離に使用



execve時の「起動元ドメイン」と「実行ファイルのタイプ」から新しいドメインを決定

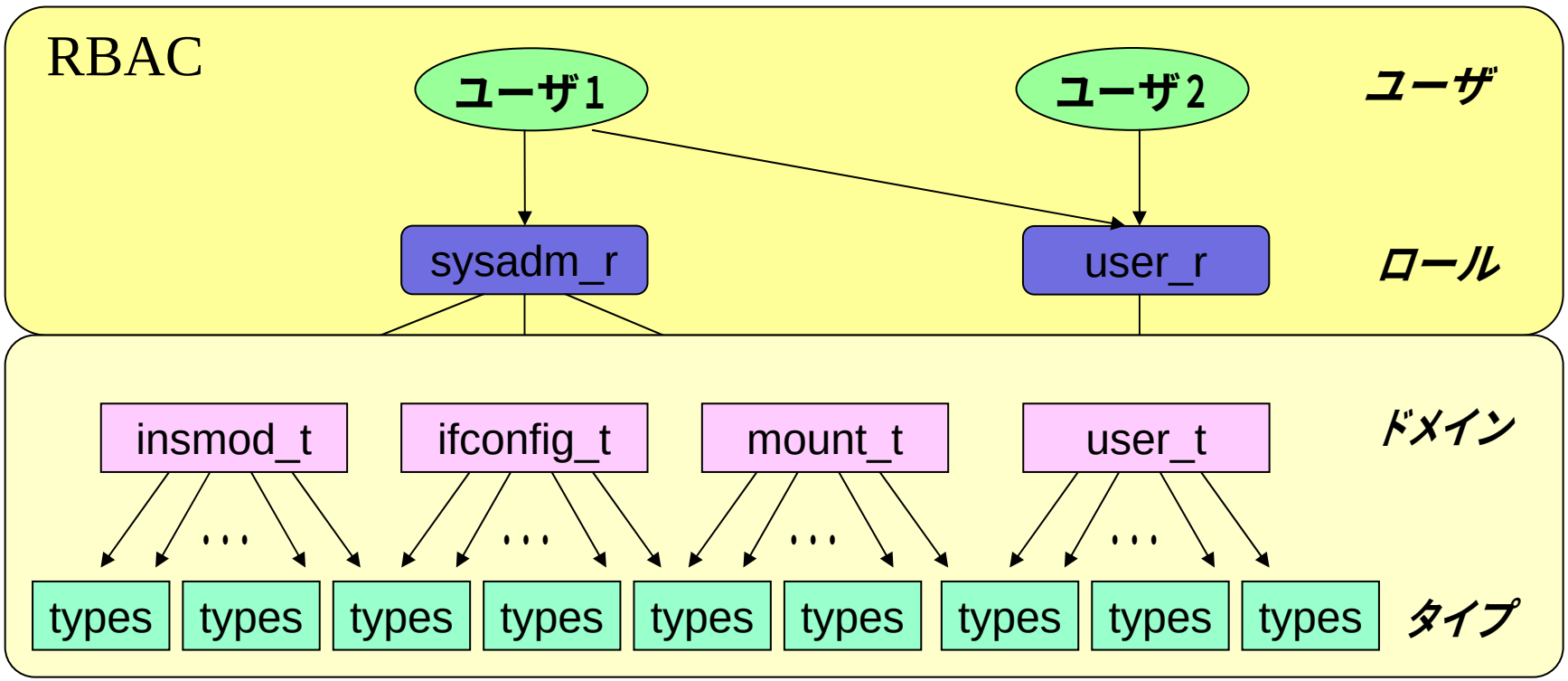
domain_auto_trans(initrc_t, httpd_exec_t, httpd_t)

＃initscriptが，/usr/sbin/httpdを起動するとプロセスは，httpd_tドメインに遷移

ドメインへのエントリポイントを制限することにより権限昇格を禁止

SELinuxのRBAC

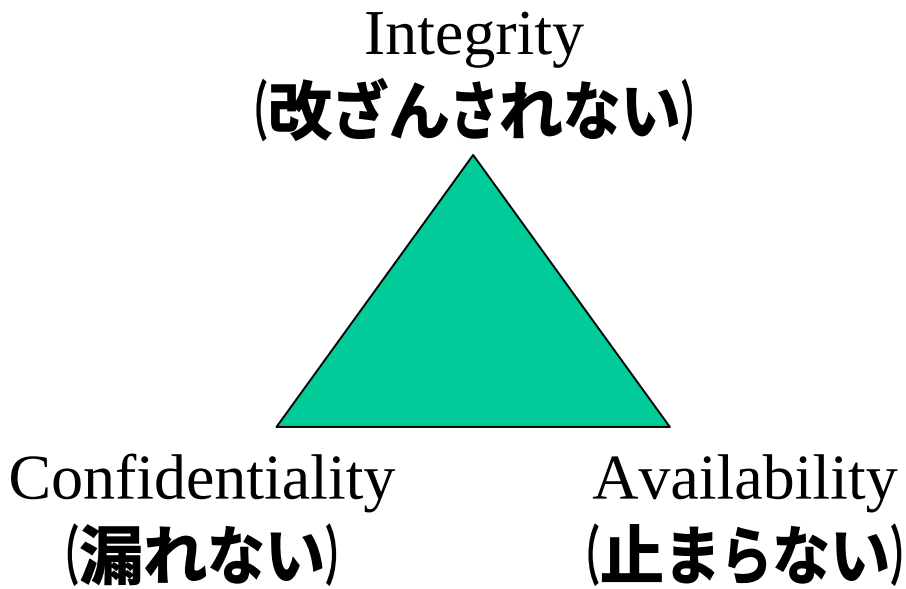
- ユーザ毎に権限を分割する仕組み
- 使用できるドメインのセットとしてロールを定義
- ユーザ毎に使用できるロールを定義



SELinuxの位置づけ

セキュリティ目標とSELinux

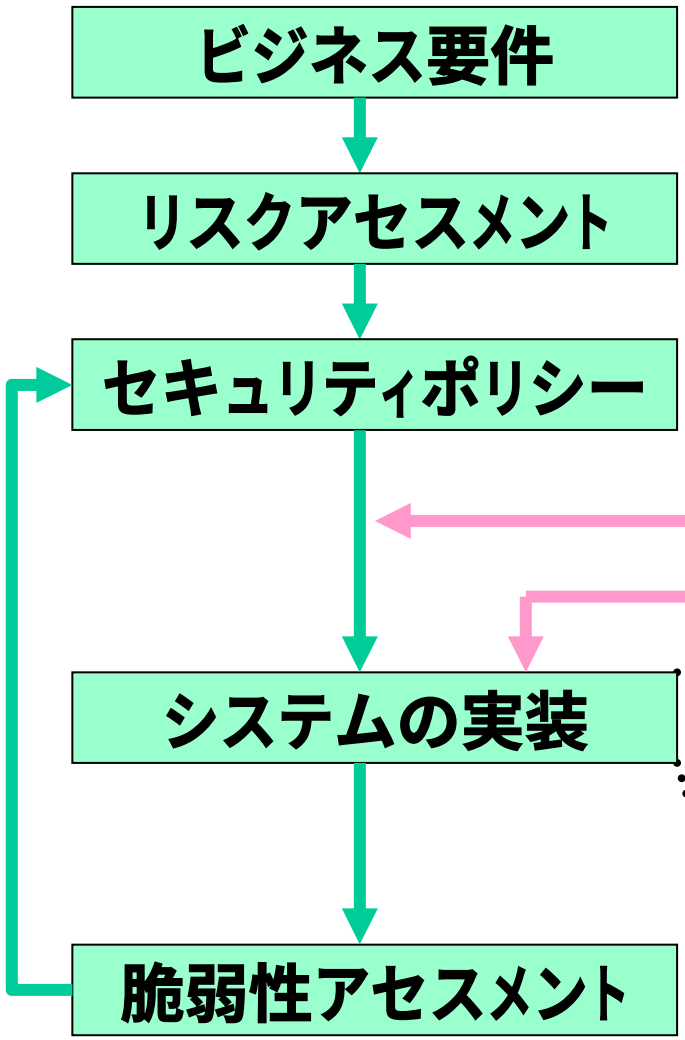
セキュリティ目標のビッグスリー



SELinuxは，ConfidentialityとIntegrityに関する技術

Availabilityに関しては，別途対策が必要

情報セキュリティマネジメントとSELinux



- セキュリティマネジメント
  - 経営者のコミット
    - ビジネスプロセス全体の見直し
    - 十分なリソース
  - 何をどのように守るのかといった ポリシー決めてシステムに反映
  - PDCAの体系的な実施

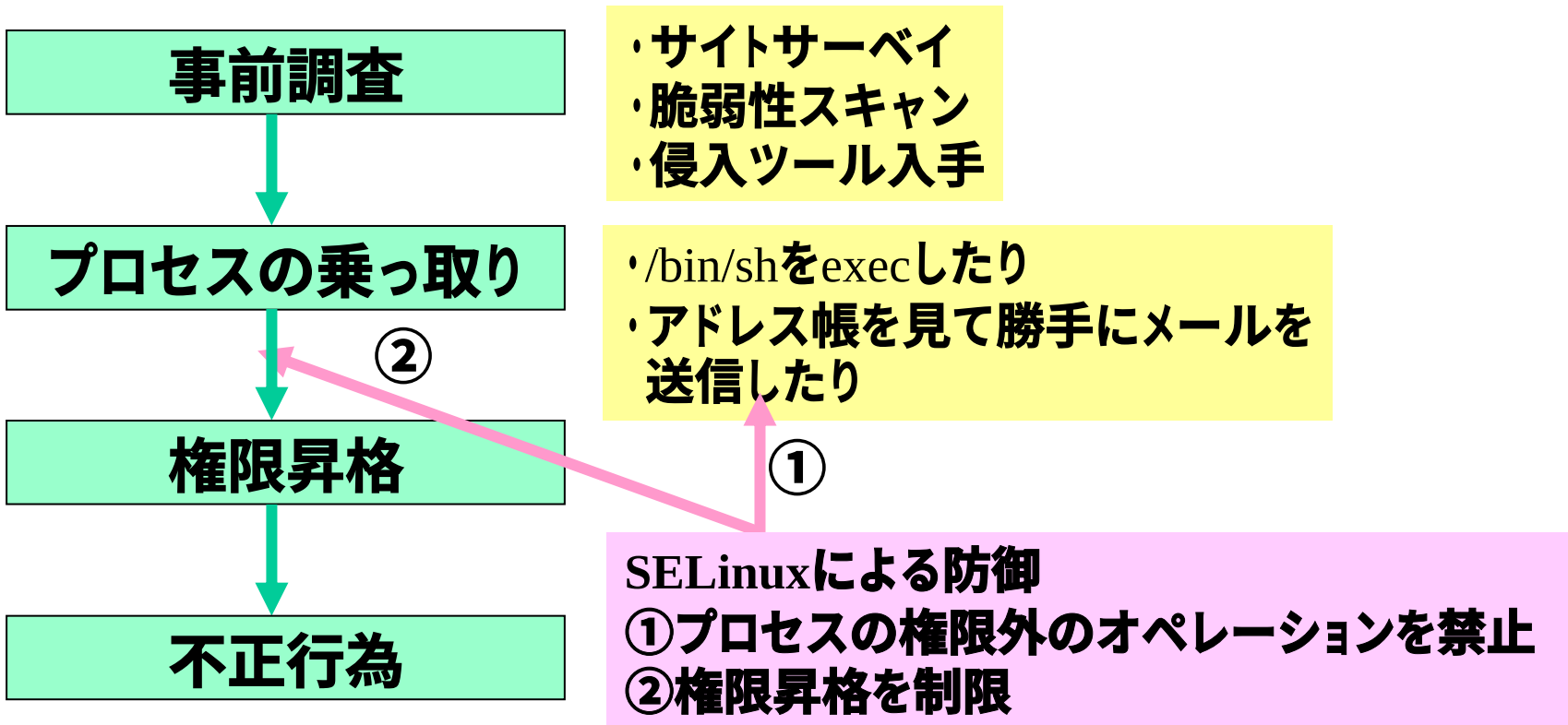
SELinuxの意義

- セキュリティポリシーの実装表現力向上
- 事故の予防（クラッキング困難化）
- 事故の被害の最小化・局限化

テクノロジサイド	ヒューマンサイド
・物理セキュリティ ・境界防御 ・エンドポイント防御 ・証拠保全	・ビジネスプロセス ・運用手順 ・事故対応手順 ・教育

侵入攻撃とSELinux

- 典型的な侵入攻撃手順



「セキュアOS」とSELinux

- 「セキュアOS」という用語に統一された定義はない
  - 強制アクセス制御モジュールを「セキュアOS」と呼ぶベンダもあり
- ユーザから見た「セキュアなOS」のあるべき姿
  - 粒度の細かい強制アクセス制御
  - クラッキング困難化 (各種バッファオーバラン防止技術等)
  - Audit Trail
  - 強力な認証, 暗号化ファイルシステム
  - セキュアなデフォルト設定
  - 迅速なアップデート, インストールメディアの最新化
  - 長期間の安定したサポート
  - ソースコードの検証
  - etc
- 「セキュアなOS」の実現には, いわゆる「ディストリビューション」のレイヤ以上での取組みが必要
- SELinuxは, 「セキュアなOS」のキーとなる要素

セキュアなOSを目指しているものの例

- OpenBSD
  - W^X, SSP, ライブラリのランダムロード, ソースコードの監査, 迅速なアップデート, セキュアなデフォルト設定, 等々
- Adamantix
  - RSBAC, その他
- Immunix (商用)
  - SubDomain, StackGuard, FormatGuard
- Red Hat Enterprise Linux 4 (商用)
  - SELinux, exec-shield
- Longhorn (商用)
  - non-stack execution, DRM系のコンテンツ保護技術, ソースコード監査, セキュアなデフォルト設定？

SELinuxの利点と限界

SELinuxの利点

- クラッキング困難化
  - スクリプトキディによる侵入やウイルス，ワーム
- 不正アクセスの被害を最小化･局限化
  - プロセスの権限を制限し，他のプロセスやリソースから隔離，権限昇格も防止
  - 万一，侵入されてもほとんど何もできない
- パッチマネジメントにおける安心･安全の向上
  - アプリケーションにシステム乗っ取りが可能な脆弱性があっても被害を制限可能
- アウトソーシングの信頼性向上
  - システムの運用は任せても，情報の閲覧は制限など

SELinuxで防止できるもの

- アプリケーションの脆弱性を利用したシステムへの侵入被害
- OSコマンドインジェクション（Web）による攻撃
- 一般的なウイルス，ワームへの感染・症状発現
  - ファイルや他のプログラムにアクセスするタイプのもの
- ユーザの権限を越えるオペレーション

SELinuxで防止できないもの

- カーネルのバグを利用した攻撃
  - 基本的に何でもできてしまう
- プロセス汚染型DoS
  - バッファオーバフロー自体は防止できない
- リソース消費型DoS/DDoS
  - システムリソース（CPU，メモリ，NW帯域，ディスク帯域，プロセス数，ファイル数等）を制限する機能はない
- プロセスの権限で完結する不正行為
  - XSS，SQLコマンドインジェクション，メール第3者中継
  - 感染プロセスのメモリ内で動作してトラヒックを流し続けるワーム

SELinuxで防止できないもの

- なりすまし
  - 認証とセッションハイジャック対策が別途必要
- 盗聴
  - 通信路の暗号化が別途必要
- 物理的な攻撃
  - ディスクの盗難や他のカーネルでブートされる等
  - 施錠，暗号化ファイルシステム等が必要
- オペレーションミスによる情報漏えい等
  - 公開ディレクトリに機密情報を置いてしまったり．．．
  - 外部メーリングリストに機密情報を流してしまったり．．．
  - 適正なマネジメントが必要

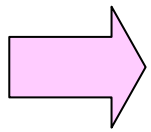
xSPにおける使い方

サービスプロバイダの課題

- ISP, IDC, ASP, MSP. . .
  - インターネットに繋ぐ
  - システム, データを預かる
  - 運用･管理のアウトソーシング
- セキュリティに関する課題･困っていること
  - 自社サービスシステムの保護
  - 顧客情報･個人情報の保護
  - 証拠保全 (ロギング)
  - 踏み台, のっとり
  - ワームによる無駄トラヒック
  - SPAM
  - パッチマネージメントの負荷
  - 等々

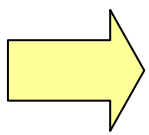
課題の分類

- マネジメントとテクノロジでいずれ何とかできそうな部分
  - 顧客情報の保護
  - 自社サービスシステムの保護
  - 預かっているシステム･データの保護
    - MSP, ASPの場合
  - 証拠保全
  - パッチマネージメントの負荷



当面SELinuxが役に立つ領域
･クラック困難化
･被害局限化

- 新たなビジネスモデル，制度等が必要なもの
  - 預かっているシステム･データの保護
    - コロケーション，専用ホスティングの場合（管理はお客様）
  - エンドユーザシステムの保護
    - 個人情報，機密情報の保護
    - ワーム感染による無駄トラヒック発生源
    - SPAM発生源

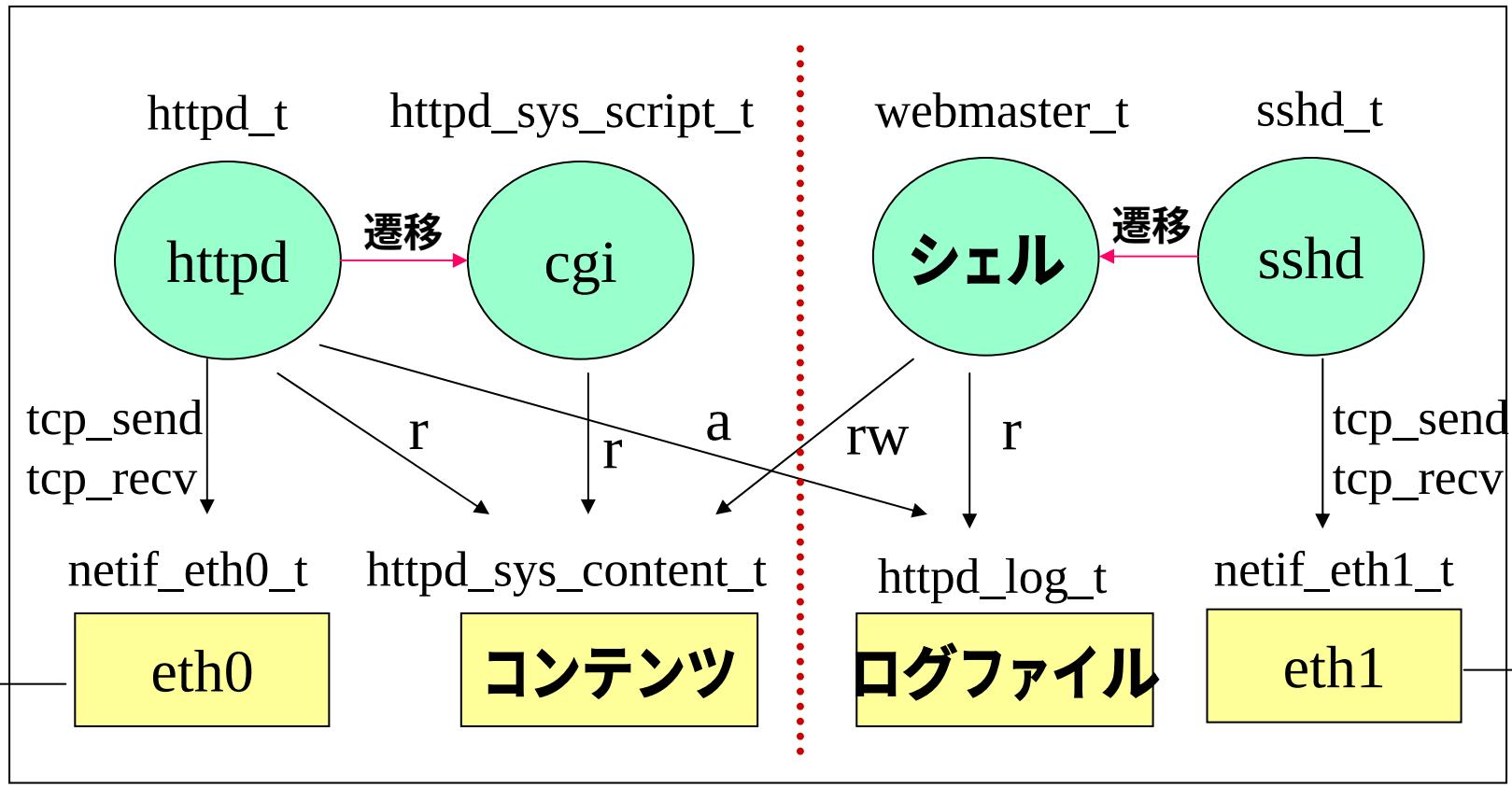


お客様に使ってもらえれば．．．
･現時点では妄想？

サービスプロバイダにおける導入対象の例

- 顧客情報を扱うサーバ
  - 申込み／問合せ受付用Webサーバ
- サービス提供の基盤となるサーバ
  - Web, DNS, Mail, News, NTP
  - SIPサーバ
  - 認証サーバ
- アプリケーションサーバ
  - ASP, 電子モール, コンテンツ配信
- ホスティングの付加サービス
  - ポリシ設定のアウトソーシングを含めて受託
- IP電話端末

ポリシー設定のイメージ



- 外部公開IFを読み書きする低信頼ドメインの権限を最小化
- 低信頼ドメインからクリティカルなリソースへのアクセスを禁止
- 低信頼ドメインから管理用ドメインには遷移を禁止

SELinuxは難しい？

- ポリシー開発は結構大変
  - ある程度のサンプルポリシーは存在する
  - 機械的な作業で書ける部分が多い
  - 難しくはないが，面倒くさい
- テンプレートを作ってしまえば，同じ種類のサーバへの設定は簡単
  - 設定ファイルをコピー
  - make load ; make relabel
- 日常の運用は，通常のLinuxと大差ない

導入に向けてやるべきこと

- 各サービスプロバイダ
  - 使い方の習得
  - テンプレートポリシーの開発
  - 運用手順，マニュアルの整備
  - 実際の利用シーンを想定した検証，トレーニング

- コミュニティ
  - サンプルポリシーの充実
  - 動作実績，ノウハウの蓄積，情報交換
  - 設定･運用簡易化ツールの充実
  - 足りない機能？の追加

SELinuxの運用・管理

SELinuxのポリシー設定

- SELinuxセキュリティポリシー
  - 上位のセキュリティポリシー (人間の願望・意思) をシステムに反映するためのアクセス制御の設定
- 実は機械的な作業で書ける部分も多い
  - プログラムが動作する必要最小限のパーミッション
  - ディストリビューションやシステム毎に固有の条件
    - インストールパス, パッケージ間の依存関係, ブートシーケンス (initscript) を反映する必要がある
- 機械的には書けない (意思を込める) 部分
  - クリティカルなリソースの定義と保護
  - 特定のIF, プログラムからReachできるドメインの制限
  - ロールに割り当てる権限, ユーザに割り当てるロール

設定項目の概要

- TEの設定
  - タイプ(ドメイン)名の定義
  - 静的なファイルに割り当てるタイプの設定
  - ネットワークポートに割り当てるタイプの設定
  - プロセスに割り当てるドメインの設定
  - システム運用中に動的に生成されるファイルに割り当てるタイプの設定
  - アクセス制御ルール(パーミッション)の設定
  - ログ監査の設定
  - 決して許可しないパーミッションの設定
- RBACの設定
  - ロールの定義
  - ロール毎に使用可能なドメインの設定
  - ユーザ毎に選択可能なロールの設定

設定方法

- テキストファイルに設定を記述
  - /etc/security/selinux/src/policy/*
- 設定言語の文法は以下のリソースを参照
  - Configuring the SELinux Policy
    - http://www.nsa.gov/selinux/policy2-abs.html
  - セキュアなインターネットサーバー構築に関する調査
    - http://www.ipa.go.jp/security/fy14/contents/trusted-os/guide.html
  - SELinux入門
    - 日経Linux 2003年9月号～2004年4月号
  - セキュリティ最前線：SELinuxを究める
    - Software Design 2004年1月号～2004年2月号

ポリシー設定のシステムへの反映

- ポリシーコンパイラでバイナリ表現に変換し，カーネルにロード
  - cd /etc/security/selinux/src/policy/
  - make load
  - ブート時は，initにより自動的に読み込み
- ファイルシステムの拡張属性にラベルを設定
  - cd /etc/security/selinux/src/policy/
  - make relabel
  - 以下の方法で特定のファイルのみラベリングも可能
    - setfiles xxx.fc dir_name
    - chcon <scontext> filename

ポリシー開発の実際

- 機械的な作業で書ける部分
  - プログラムの動作するドメインを定義
  - 定義したドメインで取りあえずプログラムを動かす
  - AVCメッセージを見て，必要なパーミッションの検討
    - straceの結果も有力な判断材料
  - ポリシー設定の追記，修正
- 意思を込める部分
  - セキュリティ等価でないものに個別のタイプを割当て
  - ドメインからタイプへの不要なアクセス権限を削除
  - 外部公開インタフェースに読み書きするドメインから重要なファイルを扱えるドメインへの遷移を禁止
  - ロールの権限を定義し，ユーザにアサイン

AVCメッセージの解析

Dec 5 20:49:09 hostname kernel: avc: denied { read } for pid=1663 exe=/bin/bash

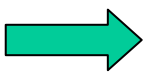
拒否されたアクセスベクタ *サブジェクトの実行ファイル名*

name=named.conf dev=03:02 ino=229451 scontext=system_u:system_r:initrc_t

オブジェクトのファイル名 *オブジェクトのinode番号* *アクセス元ドメイン*

tcontext=system_u:object_r:named_conf_t tclass=file

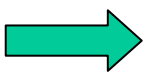
アクセス先タイプ *オブジェクトのクラス*



パーミッションの設定

allow initrc_t named_conf_t:file read;

Dec 5 21:11:02 hostname kernel: avc: denied { read } for pid=5957
exe=/usr/sbin/snmpd name=IP-MIB.txt dev=03:02 ino=8847788
scontext=system_u:system_r:snmpd_t tcontext=system_u:object_r:usr_t tclass=file



ファイルコンテキストの設定

/usr/share/snmp/mibs(/.*)? system_u:object_r:snmpd_mib_t

パーミッションの設定

allow snmpd_t snmpd_mib_t:file read;

ポリシー設定の検証

- 目視確認
  - 重要なタイプにアクセス可能なドメイン
  - 重要なドメインに遷移できるドメイン
  - 低信頼なドメインから遷移できるドメイン
  - 確認ツール
    - grep
    - setools (apol)
- assertion
  - 「許可したくないオペレーションの集合」として，「あるべき姿」(セキュリティ要件) を定義
  - ポリシーコンパイル時に違反がないかチェック
- 情報フローモデルをベースとした形式的な検証
  - あるべき姿を別の言語で定義し，違反がないか検証

オペレーション

動作モード

- permissiveモード
  - アクセス制御は実行せずにロギングのみ行う
  - ポリシー開発用
- enforcingモード
  - アクセス制御を実行
- モードの参照
  - cat /selinux/enforce
  - 0 => permissive，1 => enforcing
- モードの設定
  - echo -n 1 /selinux/enforce ：enforcingモードに切替
  - echo -n 0 /selinux/enforce ：permissiveモードに切替
- カーネル構築オプションにより，enforcingモードでのみ動作するカーネルを構築可能

基本的なオペレーション

- カレントロールの参照
  - id
- ロールの切替
  - newrole -r sysadm_r
- プロセスのセキュリティコンテキストの参照
  - ps --context / ps -Z
- ファイルのセキュリティコンテキストの参照
  - ls --context / ls -Z
- ファイルのセキュリティコンテキストの設定
  - chcon <scontext> file

基本的なオペレーション

- initscriptの実行
  - run_init /path/to/initscript
- ファイルのバックアップ／リストア
  - star -cv -xattr H=exustar f=backupfile dir/
  - star -xv f=backupfile
- パッケージのアップデート
  - rpmコマンドやinstallコマンドで通常通りアップデート
  - 更新ファイルのセキュリティコンテキストを設定
    - setfiles xxx.fc / 等でラベルの適正化が必要

基本的なオペレーション

- ユーザアカウントの追加
  - 通常通りアカウントを作成
    - useradd test
    - passwd test
  - ユーザへのロールの割当て
    - /etc/security/selinux/src/policy/usersを編集
      - user test roles { user_r sysadm_r }; など
    - ポリシーの再ロード
  - ホームディレクトリのラベリング

```
#find /home/test ¦ xargs chcon -h  system_u:object_r:user_home_t
#chcon -h system_u:object_r:user_home_dir_t /home/test
```

オペレーションにおける注意点

- パーミッション不足による不具合
  - DACだけでなくMACの設定にも気を配る必要がある
- ファイルのセキュリティコンテキスト
  - 新規生成ファイルのコンテキストは，デフォルトでは親ディレクトリのコンテキストを継承
  - 想定している設定と変わってしまう場合がある
    - テキストエディタでファイルを編集した場合
    - rpmなどでパッケージをアップデートした場合
  - タイプ遷移の設定を行うか，ファイル編集後にsetfiles/chcon
- 運用中の make relabel
  - タイプ遷移で特殊な値を付けているラベルが初期化されないように注意が必要
    - /path/to/the/dynamic/labeld/file　<<none>>
  - ttyデバイスのラベルが初期化されて，コンソールにアクセスするプログラムが動作しなくなる (ログインしなおせば問題ない)

まとめ

- SELinuxの効果
  - クラッキング困難化
  - 被害の最小化･局限化
- SELinux単独では完璧ではない
  - 他のセキュリティ対策と併用することが必要
- ポリシー開発には，ある程度のコストがかかる
  - サンプルポリシーが充実すれば随分楽になる
- サーバの日常オペレーションは難しくない
- Webサーバとして利用する場合のパフォーマンスペナルティは大きくない
- 皆で使えば楽になる