社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2003－CSEC－23　(5)
2003／12／19

OS カーネルにおける動的アクセス制御

保理江 高志　　桝本 圭　　宮本 洋輔

原田 季栄　　田中 一男

（株）NTT データ 技術開発本部

あらまし

近年、Security-Enhanced Linux（以下、SELinux）[1][2]等のセキュアOSに注目が集まる。これらはアクセス制御上の拡張をOSカーネルに実装したものである。セキュアOSには、MAC、TE、RBAC等の機能が実装されており、高いセキュリティ強度を持つが、運用上のいくつかの問題点が指摘できる。アクセス制御情報の記述における柔軟性の欠如であり、セキュリティ管理者権限が通常の稼動時においても効力を持つことに起因するリスクであり、不正アクセスが発生した場合でも、何らかの能動的なアクションに結びつける術がないという点である。

本稿ではOSカーネルをベースとした不正アクセスに対する防御（インシデントレスポンス）の試みとして、SELinuxを元に行った機能拡張に関する検討及び実装に関して報告する。

Dynamic Access Control for Operating System Kernel

Takashi HORIE　　Kei MASUMOTO　　Yosuke MIYAMOTO

Toshiharu HARADA　　Kazuo TANAKA

Research and Development Headquarters, NTT DATA CORPORATION

Abstract

In recent years, security enhanced OS such as SELinux has been focused on. Some access control extensions are implemented in its OS kernel. Security enhanced OS has several security functions such as MAC, TE, RBAC, and its security intensity is improved. But some problems with the field use of it can be pointed out. They are the lack of flexibility on writing access control rules for it, the risk that is caused by the efficiency of security administrator role during runtime of the system, and disability on dealing with illegal access reactively.

In this paper, we proposed SELinux based extensions and implementations those enabled intrusion detection and dynamic access control.

1. はじめに

現在、ネットワークセキュリティ対策の主流としてファイアウォール・IDS 等が広く普及し、一定の成果を挙げている。その一方でホームページ改ざんや情報漏えいといった不正アクセスによる被害は依然、減少する傾向が見られない。

ファイアウォール・IDS 共に予測しうる攻撃への対策であり、新規の攻撃手法に対してはほとんど効力を持たない、ということが一般には認知されていないのも一因であるが、ネットワークセキュリティ対策上の「未知の攻撃手法への対処」や「侵入後の被害の局所化」といった課題は、今後さらに重要視されると考えられる。これに対する、有効なアプローチの一つが、セキュア OS である。

これは OS カーネルに対し、強制アクセス制御（Mandatory Access Control; MAC）等のアクセス制御上の拡張を施したもので、新規の攻撃手法等により、クラッカーに侵入を許し、root 管理者権限を奪われた場合でも、不正な行為に対しては、アクセス制御レベルでそれを阻止する機能を持つ。

一方で、セキュア OS においても運用上のいくつかの課題を指摘することができる。以下の 3 点である。

- 設定上の柔軟性の欠如
- セキュリティ管理者権限に付随するリスク
- 不正アクセス検知後の対応手段を持たない

本稿では、セキュア OS に対し「不正アクセス検知」という従来 OS が持たなかった機能拡張を行い、その検知情報をトリガとして OS カーネル

が自らセキュリティ状態を制御する機構について検討し、NSA が開発したオープンソースのセキュア OS、SELinux をベースとして行った本拡張の実装[3]について報告する。

2. セキュア OS の概要

2.1 アクセス制御機能

セキュア OS のいくつかの機能を概説する。

- MAC
  アクセス権設定に関する全権は secadm（セキュリティ管理者）が保持し、その設定が「強制」される。root ユーザーであってもそれを override できない。
- Type Enforcement (TE)
  各アプリケーションのアクセス権の範囲を排他的な「ドメイン」として定義する。
- Role Based Access Control (RBAC)
  システム管理者権限を役割毎に分割し、root への管理者権限の集中を抑える。
- 監査ログ
  パーミッションチェック時の種々の情報を履歴としてログ出力させる。

2.2 SELinux におけるアクセス制御情報の記述

SELinux においては許可されるアクセスの主体、客体、メソッドの 3 つ組を例 1 の要領で記述する。

例 1　allow ステートメント

```
allow sysadm_t sysadm_home_t:file {
      create execute ioctl read getattr lock write
      setattr append link unlink rename
      };
sysadm（root 管理者）の自らのホームディレクトリ配下のファイルに対するパーミッション
```

監査ログへの出力内容については auditallow、auditdeny 及び dontaudit ステートメントにより、例 2 のように記述する。

例 2　auditallow/auditdeny/dontaudit ステートメント

```
auditallow sysadm_t kernel_t:system avc_toggle;
sysadm の、kernel タイプのリソースに対する avc_toggle の実行許可をシスログに出力
auditdeny sample_t ld_so_cache_t:file {*};
sample ドメインのプロセスの、共有ライブラリキャッシュに対する全てのアクセス拒否をシスログ出力
dontaudit httpd_admin_t boot_t:dir read;
httpd_admin ドメインのプロセスの、/boot 配下への read に関するアクセス拒否はシスログ出力しない
```

3. セキュア OS の課題

SELinux は TCSEC(Trusted Computer System Evaluation Criteria) B1 グレード相当と言われており、実運用に供しても十分なセキュリティ強度を持つとされている [4] 。だが、その効力が発揮されるためには、各システム資源へのアクセス制御情報が適切に設定されているのが前提となる。また、セキュリティ管理者権限についても、不正に利用されることのないよう、対策を講じる必要がある。本章では SELinux におけるいくつかの課題点について述べる。

3.1 モノリシックなセキュリティ設定

SELinux のアクセス制御情報ファイルには、システムの置かれるあらゆる局面での、全てのアクセス権限が記述されている。その中にはセキュリティ管理者による「アクセス制御情報の変更」等の権限も存在する。しかしこれらの権限はサービスの運用上は必要性がなく、その権限の存在はむしろリスクとなるケースも存在する。

具体的には、セキュリティ管理者権限が不正に奪われた場合のアクセス制御情報の改ざんや、悪意あるセキュリティ管理者による内部犯行等が挙げられる。アクセス制御情報はシステムリソース保護の根幹を成す部分であり、いったん確定すれば、以後頻繁に変更が行われる性格のものではない。運用時にこれらの権限が許可されていなくとも、さほど不都合はない。

これらは本来、「管理」「運用」といった局面ごとに分離されるべきところを、モノリシックなセキュリティ状態の下、共存させて定義せざるを得ないところに要因がある。サービス運用に直接必要の無い権限は極力、禁止することによりセキュリティ強度は向上するものと考えられる。その上でいかにして安全にセキュリティ管理者のオペレーションを行うべきかを検討すべきである。

3.2 安全性、利便性の両立

SELinux のアクセス制御情報ファイルに対しては、必要最小限（余分な権限が与えられていない）かつシステム及びアプリケーションの動作に十分なアクセス権設定が行われなければならない。

一方でアクセス制御情報ファイルの規模は、時に数万行にも及び、管理上もアクセス制御情報全体の把握を困難なものとしている。

このため、セキュリティ管理者のとる一つの方策として、安全サイドへのマージン、即ちパーミッションの厳格化という方向でのアクセス権設定が行われる場合がある。この場合ユーザーの利便

性が著しく損なわれる場合があり、結果的にセキュリティ管理者の意図したガイドラインが定着しないというような弊害も招き得る。利用者のためのサービスである以上は、利便性も図られるべきであり、「安全性」との両立が求められる。

しかし、前述の「モノリシックなセキュリティ状態」の下で、種々の事態を想定した上で、必要最小限かつ、両者のバランスを満たすアクセス制御情報を見出すのは非常に困難であると言える。

4. OS カーネルベースの自律防御

4.1 OS カーネルでの不正アクセス検知・遮断

不正アクセスを検知し、必要な対処を自動的に行う連携機構がいくつか提案されている。例として、IDS 等での検知をトリガとして、ファイアウォールのルール等を動的に変更し遮断する、という手法である。[5]

この機能は OS カーネル以外の部分で検討されることが多いが、リアルタイムに検知・遮断を行う場合には、以下の理由から、OS の機能として実現することに有用性があると考えられる。

ネットワーク型 IDS をベースとした場合、問題となるのが、検知から対処までのタイムラグである。特にワーム型の不正アクセスをリアルタイムで阻止しようという場合、検知直後にパケット破棄等の対処を行わないと、検知はしたものの感染という事態を招き得る。

また、アプリケーションレベルでこれを実施する場合、タイムラグに関しては解決可能であるが、検知・遮断に要するホスト側の処理負荷が問題となる。

一方、OS カーネルでの検知を行う場合、パケットは意味的に解釈され、最終的にシステムコールの形に置き換えられるため、検知の処理負荷はパケット監視よりも低減される。また、カーネルレベルでの実際の動作・システム資源へのアクセス内容を踏まえた判定が可能なため、未知攻撃に対してもある程度の適応性を示すものと考えられる。

以上のような点で OS カーネルでの本機能の実現に有用性があるものと考える。

4.2 状態遷移可能なカーネル

3 章で述べたように、システムに求められるセキュリティは、利便性の観点からも、種々の状況により、適切な強度・内容は異なる。特にサービス運用時と管理者作業時においてはその差が顕著であると思われる。

アクセス制御情報を保護する意味で、サービス運用中はその権限がシステムに存在しないのが、最も堅固な状態となり、理想的と言える。

そこでカーネルレベルでの不正アクセス検知後の対応手段として、カーネルの状態遷移について検討を行った。以下の 3 つのカーネル状態を定義した上で、それぞれ個別のアクセス制御情報に基づいて動作可能とする。

システムの置かれている状況に応じ、カーネルのアクセス制御情報を変更することで、システムの防御を図る、という機能を実現する。

- 「監査」状態
  セキュリティ管理者のオペレーション用環境、アクセス制御情報を変更可能
- 「運用」状態
  通常のサービス稼動用の環境
  アクセス制御情報の変更禁止
- 「保護」状態
  防御用の環境、アクセス制御を厳格化、一般ユーザー及び管理者（システム及び各アプリケーション管理者）権限を制限

セキュリティ管理者の権限を「監査」状態として分離することにより、サービス稼動時の状態である「運用」「保護」状態においては、いかなるユーザーもアクセス制御情報への不正な改ざんが不可能となり、「アクセス制御情報の保護」が実現される。

また、いくつかの管理オペレーション（デーモンやアプリケーション設定ファイルへの操作等）は、クラッカーによるバックドアの設置等で、利用される危険性があり、先ほどの「アクセス権設定情報の保護」と同様、「アプリケーション設定ファイルの保護」を目的として、それらの管理者に対しても、不正アクセスを受けていると見られる場合には、設定ファイル等の変更を禁止することで、「保護」状態としての定義を行う。

4.3 各セキュリティ状態／監査状態の定義

各セキュリティ状態におけるアクセス制御情報の記法として、allow、auditallow、auditdeny 及び dontaudit ステートメントへの例 3 に示す拡張を行った。

末尾にセキュリティ状態・監査状態を示す、ラベルを付与可能とした。従来のアクセス制御情報ファイルとの互換性を意図し、LABEL パラメータ省略時は、全状態でそのステートメントが有効なものとした。具体的記法を例 4 に示す。

例 3　状態ラベルの記法

allow | auditallow | auditdeny | dontaudit
<subject> <object>:<class> <perms> <LABEL>;
subject:アクセス主体のセキュリティコンテキスト
object:アクセス客体のセキュリティコンテキスト
class:オブジェクトクラス
perms:対象となるパーミッション
LABEL:状態ラベル

例 4　状態ラベルの用例

allow secadm_t seccmd_exec_t:file { getattr read execute ioctl execute_no_trans } 0;　　# 監査状態
… 監査状態での secadm の管理コマンド実行ファイルに対するパーミッション定義
allow sysadm_t httpd_conf_t:file { create ioctl read getattr lock write setattr unlink..} 1;　# 運用状態
… 運用状態での sysadm の httpd 設定ファイルに対するパーミッション定義
allow sysadm_t httpd_conf_t:file { read getattr lock ioctl } 2;　　# 保護状態
… 保護状態での sysadm の httpd 設定ファイルに対するパーミッション定義（運用状態よりも制限）

4.4 検知対象の定義

セキュリティ状態・監査状態の制御を、管理者のオペレーションや監視アプリケーションからのアラート等、外部からの手段に依存する場合、間接的にカーネル状態への干渉・不正操作が行われてしまう可能がある。このため以降で述べる方式により、カーネル自らがセキュリティ状態の制御を行い、外部からの干渉・操作を禁止する、という方針をとる。

今回検討を行ったのが、カーネル自身による不正アクセス検知をトリガとした、セキュリティ状態の制御である。監視すべき不正アクセスをアクセス制御情報への拡張により、例 5 の 2 種類のステートメントにより定義可能とした。

例 5　検知対象の定義

strict | watch
<subject> <object>:<class> <perms> <LABEL>;
※<subject> 〜 <LABEL> 内容は例 3 と同様
strict ~admin { lib_t bin_t sbin_t }:file { write append unlink rename };
… admin ドメイン以外のプロセスが、lib,bin,sbin タイプのファイルに対して{ write append unlink rename }を行った場合、セキュリティ状態を遷移
watch httpd_admin_su_t shell_exec_t:file read;
… httpd_admin_su_t ドメインのプロセスが、shell_exec_t タイプのファイルに対して read を行った場合、ログ監査状態を遷移

定義されたアクセスが検知された場合、strict の対象であればセキュリティ状態を強化し、アクセス権を縮小する。

watch の対象であれば監査状態を強化し、ログへの出力内容を詳細化する。一例として通常はアクセス拒否とされたもののみをログ出力し、監査状態が強化された場合は、アクセス許可とされたものもログ出力する、という制御が可能である。

なお、検知が行われた場合、'detected'ログを出力する。例 6 に strict ステートメントによる定義と併せて、ログ出力内容を示す。

例 6　検知ログ出力

【検知指定内容】
strict staff_t var_log_t:dir search
… staff ドメインのユーザーによる、/var/log 配下へのディレクトリ走査を検知

【検知ログメッセージ】
Nov 5 9:38:20 infini kernel: avc: **detected** { search } for pid=123 exe=/bin/cat path=/var/log dev=08:02 ino=246 scontext=root:staff_r:staff_t tcontext=system_u:object_r:var_log_t tclass=dir **slevel 1->2**
… 定義対象を検知し、セキュリティ状態が 1（運用）から 2（保護）に制御されたことを示す。

4.5 セキュリティ状態の制御

アクセス権設定情報への不正操作の阻止、「監査」状態保護の実現のため、可能なセキュリティ状態の遷移を図 1 のように制限した。

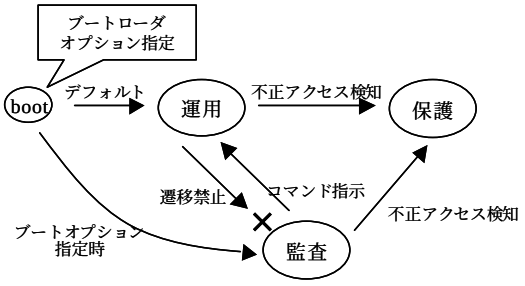

図 1　状態間の遷移

直接的なセキュリティ状態及び監査状態の指定はブート時のオプション指定でのみ可能とした。「運用」「保護」状態から監査状態への遷移を許可しないことで、セキュリティ管理者権限の保護を実現する。「監査」「運用」状態では不正アクセスの検知により、「保護」状態への遷移が発生する。これはセキュリティ管理者であっても、禁止されている操作を行った場合には保護状態への遷移が行われることを示す。

4.6 Tampering への対処

防御手段としてこの種の動的制御を行う場合に問題とされるのが Tampering、即ち故意にセキュリティ状態や監査状態を変動させることを意図した攻撃である。後述するシステムコールインタフェースを介した連携を行うような場合には、サービスへの影響を招いてしまうため、Tampering への対策が必要とされる。

今回拡張を行った strict/watch ステートメントにおいても allow での拡張同様にセキュリティ状態毎の記述を可能としているため、インシデントの危険度に応じた「運用」状態の層別化を行うことにより、Tampering の影響を軽減することが可能である。状態遷移図においては「運用」状態を図 2 のように細分化したことに相当する。

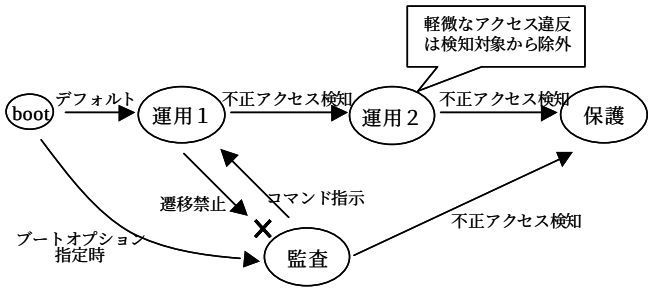

図 2　運用状態の分割

4.7 安全性判定

一定時間、strict 及び watch により指定されたアクセスが検知されなかった場合には、安全性が確認されたとして、図 2 の「運用 2」状態から「運用 1」状態への遷移を行い、セキュリティ状態を緩和する、という制御を行った。ただし安全性判定においても「保護」状態から「運用 2」状態及び「監査」状態への遷移は、行われない。

5. 実装

5.1 SELinux への拡張

SELinux をベースに、以上の機能拡張を実装した†。主要な変更箇所は以下の通り。

- パーミッション構造体（access vector）
  - strict/watch メンバの追加
  - パーミッション情報の配列化（状態毎）
- アクセス制御関数
  - strict/watch 検知対象との比較判定ロジックの追加
  - 検知の際のログ出力（'detected'）
  - セキュリティ状態毎のパーミッション情報の切り換え
  - 監査状態毎のログ出力条件の切り換え
- 状態遷移関数の追加
  - 検知に基づく状態遷移制御ロジック
  - タイマによる安全性判定
- ツール拡張
  - 設定ファイルのパーザ及びコンパイラの拡張、入出力関数の変更
- システムコールインタフェース追加
  - 現在のセキュリティ状態／監査状態の取得（参照のみ）
- カーネルへのブートパラメータ追加
  - 初期セキュリティ状態
  - 初期監査状態
  - 安全性判定待機時間

5.2 secadm ロールの定義

「監査」状態において、アクセス権設定などの操作を行うセキュリティ管理者（secadm）の権限を以下の範囲で規定し、secadm ドメインとして定義した。これらは「監査」状態（セキュリティ状態ラベル 0）のみで allow される。対応するコマンド、許可したパーミッションについて、代表的なものを例 7 に示す。

- ポリシーファイル（ASCII）の編集（/etc/security/selinux 配下への write 等）
- ポリシーファイルバイナリの生成（checkpolicy コマンドの execute、ポリシーファイルバイナリへの write 等）
- ポリシーファイルバイナリの読み込み（load_policy コマンドの execute 等）
- オブジェクトへのラベル付与・変更（setfiles コマンドの execute 等）

例 7　secadm の権限定義

```
allow secadm_t seccmd_exec_t:file { read getattr
execute ioctl execute_no_trans } 0;
  … セキュリティポリシー操作コマンドのパーミッション
allow secadm_t setfiles_exec_t:file { read getattr
execute ioctl } 0;
  … setfiles コマンドのパーミッション
```

† ベースカーネルはlsm-2.4-SELinux-2003071106 (kernel 2.4.21)

5.3 不正アクセス検知対象の定義

strict 及び watch ステートメントの具体的な用法について述べる。strict による不正アクセス検知においては、基本的に、アクセスが許可されていないものが対象とされる。strict の検知対象としては以下のようなものが挙げられる。

- 不正な shell 実行
  通常、ftpd、bind 等のサーバープロセスから shell が起動されることは無く、バッファオーバーフロー等により不正に起動された可能性が高い。

こういった不正アクセスを行おうとするクラッカーが「踏む」可能性のあるセキュリティコンテキストは有効な検知対象である。

一方、watch の検知対象としてはインシデントではないものの、注意を要するアクセスが該当する。このため strict とは異なりアクセスの許可・不許可に関わらず、検知対象となり得る。

- 重要な管理コマンド
  所定の管理者に対してアクセスが許可されていても、極めて重要なオペレーションが行われる場合に、管理者のオペレーションログ記録として、ログ粒度を細かく制御するのは有効であると考えられる。
- 権限の無いコマンドの実行
  strict の場合ほど危険性がないもの。注意を要するが、能動的なレスポンス（セキュリティ状態制御）を行うのは過剰と見られる場合に、監視強化のみにとどめるのが妥当と思われる。

5.4 システムコールインタフェースによる連携

外部からのセキュリティ状態／監査状態操作を禁止した一方で、現在のそれらの値を参照するためのシステムコールインタフェースを追加した。以下の形のアプリケーション連携において有用と思われる。

- 侵入検知ツールとの連携
  侵入検知ツール側からカーネルのセキュリティ状態を参照することで、それらのツールでは検知不可能な、システムコールレベルのインシデント検知情報を参考情報として利用することが可能となる。
- コンテンツ／サービスレベルの制御
  動的 Web ページのスクリプト等からセキュリティ状態を参照し、それに応じて、サービスメニューやコンテンツの一時的停止・変更等の制御を行うことが可能となる。ただし、ユーザー側からサービスのデグレードと見られないよう、配慮が必要である。
- アプリケーションログの粒度制御
  アプリケーション側から監査状態を参照し、それに応じて、カーネルログ粒度制御と同様に、アプリケーションログの粒度変更等の制御を行うことが可能となる。

6. まとめ

カーネルをベースとしたインシデント検知手法、及びその情報に基づきセキュリティ状態・監査状態を自ら制御可能な OS カーネルに関して提案を行った。今後は実サービス適用を想定し、以下の課題点に関する検討を進める。

- 検知の高度化
  不正アクセス検知を、その時点までのシステムコールの文脈情報も用いて行うことで、より精度の高い検知が可能になると見られる。Tampering 対処においても有効な手段の一つであると思われる。
- 状態制御の高度化
  状態遷移の制御に関して、検知内容に応じた分岐を可能とすることで、より適切な対処に結びつけることが可能と考えられる。
  また、安全性判定においてもタイマ以外の判定項目の検討が必要である。

参考文献

[1] NSA Security-Enhanced Linux ホームページ
http://www.nsa.gov/selinux/

[2] C. Wright et al., “Linux Security Modules: General Security Support for the Linux Kernel” , USENIX Security Symposium ’02

[3] 保理江、原田、田中"アクセスベクタ拡張による Linux カーネルの自律防御"電子情報通信学会ソサイエティ大会、2003、A-7-1

[4] IPA 調査報告「オペレーティングシステムのセキュリティ機能拡張の調査」
http://www.ipa.go.jp/security/fy13/report/secure_os/secure_os.html

[5] JNSA WG 活動報告「ダイナミックディフェンスの概要と適用について」
http://www.jnsa.org/active00_8.html