SELinuxポリシ処理系のユーザ空間実装

滝澤 峰利[1,a)]　橋本 正樹[1]　辻 秀典[1,2]　田中 英彦[1]

概要：SELinux はシステム侵害後の被害拡大を OS レベルで抑止できる点で有用であるが，ポリシ記述の簡単化はもとより，ポリシの可読性や保守性，拡張性の向上が課題となっている．本研究では，それらの課題解決を支援するために，SELinux のポリシ記述方式やアクセス制御モデルの変更を簡単化する方式を提案する．提案方式は，従来カーネル空間にある SELinux のポリシ処理系をユーザ空間に移行するものであり，本稿では，その評価として，SELinux のポリシに関する研究を複数取り上げ，各々が必要とする Linux カーネルの変更項目数を，提案方式を利用した場合と比較することで，提案方式がそれらの実装を簡単化できることを確認する．また，提案方式の上で動的なアクセス制御モデルを試験的に実装し，容易に効果検証できることを確認する．

キーワード：SELinux，ポリシ処理系，動的アクセス制御

User Space Implementation of SELinux Authorization System

Minetoshi Takizawa[1,a)]　Masaki Hashimoto[1]　Hidenori Tsuji[1,2]　Hidehiko Tanaka[1]

Abstract: SELinux is an effective MAC system for preventing the damage from spreading after security breaches, and there are many challenges around its policy processing issues such as readability, maintainable and scalability. In this study, we propose a basic system to facilitate experimental replacements of access control model and policy description language of SELinux in order to tackle policy processing issues, by migrating the authorization system of SELinux from kernel space to user space. As for the evaluation of our system, we confirm that it reduces the workload and difficulty of implementing some related works so far, and it facilitates an implementation and verification of trial dynamic access control model.

Keywords: SELinux, Authorization System, Dynamic Access Control

1. はじめに

1.1 研究の背景と課題

SELinux[1] は強制アクセス制御の Linux に対する実装であり，米国家安全保障局を中心としたオープンソースコミュニティによって活発に開発が進められている．強制アクセス制御は，ポリシで定めた権限を，特権プロセスを含む全プロセスにシステムコールレベルの粒度で強制するため，システム管理者は，制御対象のプロセスに細粒度の権限のみを与えることが可能となり，結果として，システム侵害後の，権限昇格等を用いた被害拡大を OS レベルで抑止可能となる．しかし一方で，細粒度のアクセス制御を行うためには，その制御規則となるポリシについても細粒度に正しく記述する必要があるが，これは人間にとって非常に困難な仕事であり，SELinux がその有用性にも関わらず必ずしも効果的に利用されているとは言い難い現状の原因となっている．

そのため，ポリシ記述の複雑性を根源とする，管理の困難性や拡張性の乏しさといった SELinux の欠点が長い間指摘され，ポリシ解析や検証に関する研究 [2], [3], [4], [5] や不要なポリシを減らす研究 [6], [7], [8]，ポリシの新たな記述方式の提案 [9]，動的なポリシ変更を行うための SELinux

[1] 情報セキュリティ大学院大学
Institute of Information Security
[2] 株式会社情報技研
Institute of Information Technology
[a)] mgs103502@iisec.ac.jp

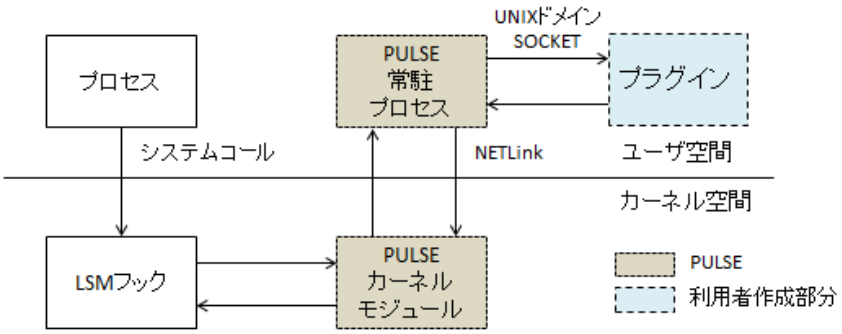

図 1 PULSE の構成
Fig. 1 Architecture of PULSE

の改良 [10] 等，その改良のために多くの研究が行われてきた．これら諸研究のいくつかは Linux カーネルの修正を必要とするが，一般に，カーネルコードのプログラミングは，資源管理の厳密さやマルチプロセスの扱いの必要性，ライブラリの欠如等の理由から難易度が高く，また，修正のたびにカーネルイメージを再構築する必要もあり，その実装と効果検証に, 研究の核以外の多大な労力を要する．

1.2 先行研究

PULSE[11] は，アクセス制御に係る認可判定をユーザ空間の常駐プロセスで行う LSM 実装 [12] で，オーストラリア国防省の A.P.Murray らによって提案された．

この研究の目的は，モジュール化された動的なセキュリティフレームワークを Linux に提供することで，従来カーネル空間で動作するように設計される LSM 実装について，アクセス制御機構の一部をユーザ空間のコンポーネントで行うことを可能とする．PULSE は**図 1** のような構成をとっており，カーネル空間とユーザ空間間の情報授受を，NETLink ソケット [*1] で行うように設計されている．その上で，認可判定を行うプラグインを常駐プロセスとは別プロセスとして構成した上で，その間のインターフェースに UNIX ドメインソケットを使用する実装となっている．PULSE は，この実装によって，管理者・ユーザ双方にインタラクティブで動的なアクセス制御の基盤を提供しており，この基盤の上で，ウィンドウシステムを利用した認可判定機構を容易に実装できることが実証されている．

PULSE は汎用的な方式であり，SELinux と連携するポリシ処理系をユーザ空間に実装することも原理的には可能であるため，これを利用することで，Linux カーネルの修正や再構成なしに，SELinux のポリシに対する新しい試みを実装し，効果検証する環境をユーザ空間に構築できる可能性がある．ただし，この場合には，SELinux に関連するカーネル内実装の全てをユーザ空間で新規に構築して PULSE と接続する構成となるため，SELinux 専用のライブラリやツール群を含めた大規模な独自開発が必要となる．

1.3 本研究の貢献

本研究の貢献は，SELinux のポリシ処理系に関する課題解決を支援するために，ポリシ記述方式やアクセス制御モデルの変更を簡単化する方式を提案し，その効果を実証することである．

提案方式は，SELinux の主たるカーネル内実装を維持したまま，ユーザ空間に実装したポリシ処理系と連携するように設計されているため，SELinux に用意されているライブラリや専用ツール群，カーネル内実装をほぼそのまま利用可能である．同時に，ポリシ処理系をユーザ空間に実装することで，SELinux のポリシに関する課題解決に向けた試みについて，Linux カーネルの修正や再構成なしに容易に実装し，効果検証できる環境を構成できる．提案方式により，SELinux の欠点を補う改良や機能拡張の研究が容易に行えるようになり，結果として，SELinux がより安全で利用しやすいものになることを期待するものである．

1.4 本稿の構成

以降，第 2 章では，本研究の関連研究として，SELinux のポリシに関する研究を複数取り上げ，続く第 3 章で，提案するポリシ処理系についての概要と実装方針，その応用としての動的アクセス制御機構について説明する．その後，第 4 章では，本研究の評価として，3 つの関連研究と第 3 章で示した動的アクセス制御機構を取り上げ，提案方式を利用して実装する場合と利用しないで実装する場合のソースの変更項目数を比較し，提案方式が新しいアクセス制御モデルやポリシ記述方式の実装・検証を簡単化できることを確認する．最後に，第 5 章で，本研究をまとめる．

[*1] Linux マニュアル セクション 7 netlink

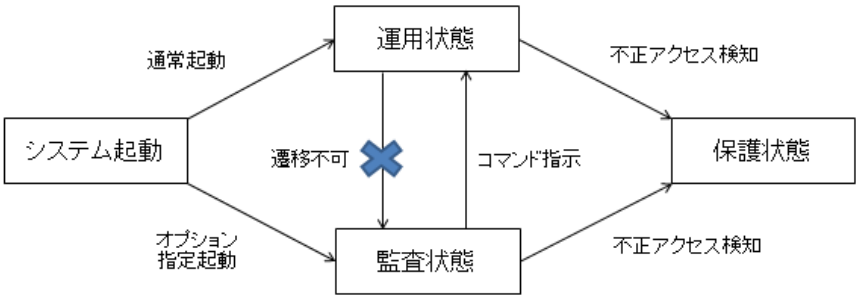

図 2 関連研究 [10] における状態遷移
Fig. 2 State Transition of Related Works[10]

2. 関連研究

2.1 不要なポリシを減らす研究

山口らの研究 [6] では，汎用的なセキュリティポリシを使用することにより与えられる，必要以上の権限を許可するポリシを自動的に発見し，システム管理者に削除を提案する手法を提案している．

この提案では，運用中実際に参照されたポリシを収集する必要があるが，auditallow [*2] による監査システムでのログ収集では，ログが大量に出力され消失する恐れがあるため，カーネルソースを変更する手法を提案している．

具体的にこの手法は，カーネルソース内のアクセスベクタ [*3] に使用済みフラグを追加し，定期的に使用済みフラグが設定されたポリシを selinuxfs ファイルシステム [*4] を通してユーザ空間に吸い上げるというものである．

しかし，矢儀らはこの提案の評価段階において，調査を行いたいポリシのみに auditallow を適用し全体のログ出力を抑える手法を取った [7]．この方法では全体のポリシについて調査する場合，ポリシの auditallow の設定を変更しながら繰り返し実験を行う必要がある．

山口らの提案手法ではカーネルソースの変更が必要になり，かなりの工数が見込めるため評価段階において上記のような手法を選んだものと考えられるが，それでもログの消失がないことを確認できる範囲で auditallow を設定し，繰り返し実験を行う必要があるため，多くの労力を要したものと考えられる．

2.2 ポリシ記述方式を変更する研究

著者らによる研究 [9] は，SELinux のポリシを論理プログラミング言語で記述しようとする試みである．論理プログラムとしてアクセス制御規則を記述することで，属性の継承や認可手順のサブルーチン化をサポートするポリシ記述言語を提案し，これによってポリシ記述の複雑さの問題を，柔軟な構造化記述手法によって解決することを目的としている．

この研究では，最終的に論理プログラミング言語の機能をカーネル空間で実装することを検討しており，その際にはカーネル内実装のためのコーディングが相当量必要であると考えられる．しかしながら，事前にアクセス制御モデルやポリシ記述言語の効果を容易に検証するための基盤があれば，研究にかかる実装工数の大幅な削減が期待できる．

2.3 動的アクセス制御を行うための研究

保理江らは SELinux のカーネルに不正アクセスを検出した場合に状態遷移を行う仕組みを実装した [10]．システムの状態には監査状態，運用状態，保護状態の 3 つの状態があり，不正アクセス検出時，運用状態から自動的に保護状態へ状態遷移を行う (**図 2**)．

このシステムの実現のために，アクセスベクタへのメンバ追加等，13 項目の変更が行われたとされているが，これらのうち 12 項目はすべてカーネルソースに影響のある変更項目であるため，多くの実装工数が必要であったと考えられる．

3. ポリシ処理系のユーザ空間実装

3.1 実装の概要と方針

本研究の目的は，カーネル内実装の変更が必要になるような SELinux のポリシ処理系に関する研究について，その実装や効果検証を容易に行えるようにすることである．そのため，提案方式では，SELinux のカーネル内実装のうち，ポリシを参照し，認可判定の元となるアクセスベクタを計算する部分をユーザ空間に抽出し，研究者がポリシ処理系を容易に変更できる仕組みを提供する．

実際の利用シナリオとしては，第 2 章で挙げたような認可判定方法やポリシ参照方法を変更する研究での使用を想定しており，SELinux のツール類やコマンド・ライブラリ

[*2] 許可した操作を記録するためのポリシに対する設定
[*3] カーネル内にキャッシュされる，SELinux による判定結果を保持する構造体
[*4] SELinux で使用される，ユーザ空間からカーネル空間の資源へアクセスするためのファイルシステム

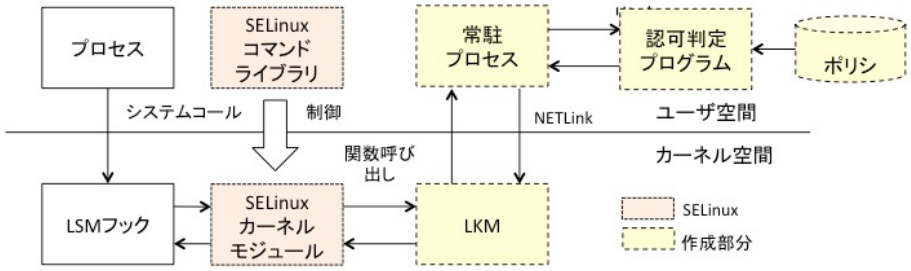

図 3 提案方式の構成
Fig. 3 Architecture of Proposed System

類には変更を加えないものとする．

提案方式は，SELinux のアクセスベクタ計算処理をフックしながらユーザ空間と通信するローダブルカーネルモジュール (Loadable Kernel Module：以降 LKM) と，ユーザ空間でその通信相手となる常駐プロセスを基礎として構成される．従来の SELinux では，認可判定処理が呼び出されると，はじめにアクセスベクタのキャッシュを調べた後に，アクセスベクタ計算関数を呼び出す仕組みになっており，提案方式では，このアクセスベクタ計算関数をフックした上で，問い合わせの元となる情報を NETLink ソケットを通じてユーザ空間の常駐プロセスに渡し，常駐プロセスのインターフェースを通じて，認可判定処理プログラムが計算を行う設計となっている．

本章では，**図 3** に示す提案方式の構成について，関連するコンポーネントである SELinux カーネルモジュールと LKM，常駐プロセス，認可判定プログラムについて説明する．また，提案方式の有効性評価のための，提案方式を基礎とする動的アクセス制御に向けた拡張実装についても説明する．

3.2 提案方式の構成

3.2.1 SELinux カーネルモジュール

SELinux の認可判定処理は，アクセス主体のセキュリティ情報，アクセス対象のセキュリティ情報，アクセス対象のセキュリティクラスから，対応する許可操作の集合であるアクセスベクタを計算することで実現される．この計算は，SELinux カーネルモジュール内の関連関数である context_struct_compute_av (以降 CAV) が担っており，この関数は以下のように定義されている．

```
/*
 * Compute access vectors based on a context
 * structure pair for * the permissions in a
 * particular class.
 */
static void context_struct_compute_av(
    struct context *scontext,
    struct context *tcontext,
    u16 tclass,
    struct av_decision *avd)
```

scontext はアクセス主体のセキュリティ情報，tcontext はアクセス対象のセキュリティ情報であり，カーネル内で定義されている context 構造体の形式で渡される．tclass はアクセス対象のセキュリティクラスで unsigned short 型である．avd は，結果のアクセスベクタを格納するための領域であり av_decision 構造体になっている．この構造体はカーネルソース中以下のように定義されている．認可判定の計算結果となるアクセスベクタの情報は，この構造体の allowed, auditallow, auditdeny に設定されて返される．

```
struct av_decision {
    u32 allowed;
    u32 auditallow;
    u32 auditdeny;
    u32 seqno;
    u32 flags;
};
```

提案方式は，CAV をフックし，認可判定の計算結果にユーザ空間から介入できるようにする．すなわち，提案方式は，この関数の代替関数を用意し，SELinux が CAV の代わりにその関数を呼び出すよう変更を加える．SELinux がこの関数を呼び出すと，ユーザ空間に用意された常駐プロセスを通して，認可判定プログラムに判定リクエストが渡され，計算結果として返されたアクセスベクタが SELinux カーネルモジュールへ返される．

3.2.2 Loadable Kernel Module

提案方式における LKM は，SELinux と常駐プロセスの仲介を行うコンポーネントである．LKM は，CAV の代替関数をカーネルに実装するもので，代替関数は SELinux から渡されたセキュリティ情報とセキュリティクラスを NETLink ソケットを通してユーザ空間の常駐プロセスに渡し，常駐プロセスから戻された計算結果を CAV と同様

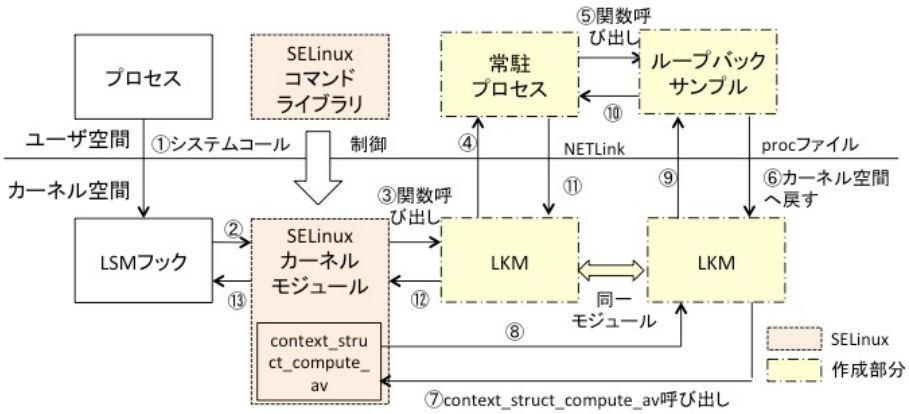

図 4 ループバックサンプルの構成
Fig. 4 Architecture of Loopback Sample

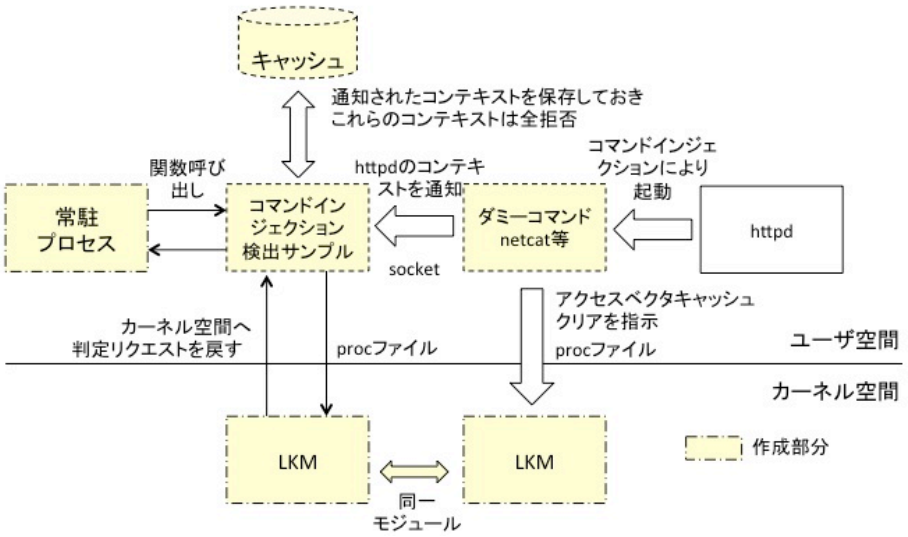

図 5 CI を検出する拡張実装の構成
Fig. 5 Extended Architecture for CI Detection

な形式で SELinux に返す．ただし，常駐プロセスを含む一部のプロセスから認可判定要求がある際に，デッドロックに陥ることが予見できるため，この対策として，ハングアップの原因となるようなプロセスのリクエストは LKM 内で例外的にすべて許可するように設計している．具体的には，常駐プロセスについては常に許可とし，加えて，Linux カーネルからデッドロックの原因となる対象プロセスのプロセス ID またはセキュリティコンテキストを取得し，LKM 内ですべてのアクセスを許可するように設計した．このようなプロセスには，sched(プロセス ID0) や init(プロセス ID1) 等があることが実験的にわかっている．

LKM は，常駐プロセスとの仲介以外にも提案方式の利用上必要となる諸機能を持ち，これらの機能は，ユーザ空間のプロセスがカーネル空間の資源にアクセスするために Linux が用意している proc ファイルシステム [*5] を通して呼び出せるようになっている．例えば，動的にポリシを変更するような SELinux の改良は，ポリシ変更時にカーネル内にキャッシュされたアクセスベクタをクリアする必要があるが，この操作は proc ファイルを通してユーザ空間のプロセスから指示できるように設計した．

また，LKM は，常駐プロセス起動時，常駐プロセスのプロセス ID を NETLink ソケットを通して受け取り，これは以降の利用に備えて LKM 内の変数に保存する．

3.2.3 常駐プロセス

常駐プロセスは，LKM から NETLink ソケットを通してアクセス主体とアクセス対象のセキュリティ情報，アクセス対象のセキュリティクラスを受け取り，あらかじめロードされた動的リンクライブラリの認可判定関数を呼び出した上で，結果として返されたアクセスベクタの情報をカーネルモジュールへ返す．

SELinux は，常駐プロセスの起動が確認された場合，SELinux 内の関数 CAV の代わりに，LKM の代替関数を呼び出す．常駐プロセスの起動は，LKM 内に保持された常駐プロセスのプロセス ID を参照して確認する．

[*5] Linux マニュアル セクション 5 proc

3.2.4 認可判定プログラム

提案方式を基礎として動作する認可判定処理の実装を含むプログラムであり，本方式の利用者が用意する．利用者は，提案方式が用意した Linux 上の動的リンクライブラリを利用してこのプログラムを作成する．

3.3 提案方式を基礎とする拡張実装

本稿では，提案方式本体とは別に，この有効性を検証するループバックサンプルと，動的アクセス制御を行うサンプルの 2 つの拡張実装を提供する．

ループバックサンプルは，提案方式本体に含まれる LKM を通して，受け取った判定リクエストをそのままカーネル内の CAV へ戻し，その結果を常駐プロセスへ返す．ユーザ空間からカーネル空間へ判定リクエストを戻す部分は，提案方式が用意する proc ファイルを通して行われ，この proc ファイルの LKM が CAV を呼び出す設計となっている (**図 4**)．この拡張実装は，山口らの研究が提案する，ログ収集のためのカーネルソースの変更や，保理江らの研究が提案する，あるインシデント発生まで通常運用を行うような研究を再現するために用意する．

動的アクセス制御を行うサンプルは，netcat 等のダミーコマンドを用意し，それらが起動されることにより httpd のコマンドインジェクション (以降 CI) を検出し，httpd に与えられたセキュリティ情報について，そのアクセス主体の認可判定をすべて拒否に変更する拡張実装である．これより，動的アクセス制御を行う場合の例を示し SELinux で動的なアクセス制御を行うための変更が，提案方式上で容易に実装・検証できることを示す．

この拡張実装では，あらかじめ用意したダミーコマンドが起動されると httpd プロセスのセキュリティ情報を socket を通して認可判定プログラムへ渡し，proc ファイルを通して提案方式の LKM にアクセスベクタキャッシュのクリアを指示する．この拡張実装における認可判定プログラムは，先に説明したループバックサンプルを元に修正して実装する．このプログラムは判定リクエストを受け取ると，まず，socket にセキュリティコンテキストが送られているか確認する．送られている場合それを読み取りメモリ上にキャッシュする．その後，判定リクエストを LKM に渡すが，アクセス主体のセキュリティ情報がキャッシュに含まれる場合，戻ってきたアクセスベクタに対し，すべての操作が拒否となるように変更するよう設計する (**図 5**)．

4. 評価と考察

4.1 評価方法

提案方式の評価は，SELinux のポリシ処理系に関する先行研究と，本稿で説明した拡張実装について，カーネル内実装の変更項目数を列挙し，提案方式を利用した場合の変更項目数と比較することで行う．その際，各変更項目が，カーネル空間におけるものか，ユーザ空間におけるものかについても注目して比較を行う．尚，ここでの変更項目とはそれぞれの論文の中で挙げられているものをそのまま採用している．

具体的には，第 2 章で説明した，「不要なポリシを減らす研究」の参照されたポリシを抽出するための変更，「ポリシ記述方式を変更する研究」の評価システムでの変更，「動的アクセス制御を行うための研究」の変更，第 3 章で説明した CI 検出のための拡張実装を対象に，提案方式を利用しない場合とする場合の変更項目数を比較する．また，CI 検出のための拡張実装については，提案方式を利用して正しく動作することも実験により確認する．

4.2 変更項目数の比較

カーネル内実装の変更項目数について，比較結果を**表 1** に示し，以下でその内容を説明する．

はじめに，SELinux の不要なポリシを減らす研究では，使用済みフラグ記録部，使用済みポリシ保存機能の 2 項目が，カーネル内実装に対する変更として挙げられている．提案方式を使用する場合，本方式が提供するループバックサンプルを変更して，参照されたポリシをファイルに記録するようにする．この研究で提案している方法では，許可されたポリシの許可ビットを記録するが，提案方式ではリクエストの許可あるいは拒否，参照された許可ビットは取得できない．許可あるいは拒否されたかどうかは，auditdeny [*6] により記録されるログと比較することで判断ができると考えられる．また，提案方式では参照された許可ビットは記録しないため，完全にこの研究を再現するものではないが，そのポリシに対して許可のための参照が行われたことは検出することが可能である．また，この研究の提案手法では，削除をポリシ単位で行うため，完全に再現せずともある程度の検証が提案方式の上で可能であると考えられる．従って，この研究を提案方式を用いて実装・検証する場合，本来の実装を再現しないとはいえ，ポリシ単位で許可参照を記録することが可能であり，2 項目挙げられていたカーネル内実装に対する変更は必要なくなる．

次に，SELinux のポリシ記述方式を変更する研究では，SELinux の認可判定関数呼び出しの変更，新規カーネルモジュールの作成の 2 項目が必要な実装項目として挙げられる．これらの変更は，提案方式とまったく同じであるため，カーネルソースの変更はすべて不要となる．認可判定プログラムにおいてリクエストを必要な形式に変換し，問い合わせ結果を SELinux の形式に変換するだけでよい．従って，この研究を実装・検証環境として提案方式を利用する場合，カーネル内実装の修正は必要なくなる．

SELinux で動的アクセス制御を実現する研究に関しては

[*6] 拒否した操作を記録するためのポリシに対する設定

表 1 変更項目数の比較
Table 1 Comparison of Numbers of Modifications

研究	カーネル空間 ユーザ空間	方式使用無	方式使用有
不要なポリシを減らす研究 [6]	カーネル	2	0
	ユーザ	0	2
ポリシ記述方式を変更する研究 [9]	カーネル	2	0
	ユーザ	1	1
動的アクセス制御を実現する研究 [10]	カーネル	12	0
	ユーザ	0	12
	不明	0	1
CI を検出する拡張実装	カーネル	2	0
	ユーザ	1	2

変更項目に関する情報が不足しているため，おおよその項目数となるが，12 項目挙げられているカーネル空間の変更の中で，12 項目すべてがユーザ空間で実装可能になると考えられる．

最後に，SELinux で CI を検出する拡張実装では，同等の機能を提案方式を使用せずに実装する場合の変更項目が，SELinux の認可判定関数呼び出しの変更，新規カーネルモジュールの作成の 2 項目となるがこれらは必要なくなる．

変更項目数の比較結果として，評価に用いた諸研究と CI 検出の拡張実装では，提案方式が大幅にカーネル内実装の変更箇所を減少し，変更箇所をユーザ空間に移行可能であることが分かった．

4.3 CI 検出拡張実装の動作検証

CI 検出拡張実装が正しく動作することを確認するため，実際に CI を行うための入力画面 cmdinj.html(**図 6**) と，ブラウザからの入力を引数に system 関数を呼び出す CI が可能な php スクリプト cmdinj.php(**図 7**) を用意して動作検証を行った．具体的には，以下の手順により実験を行った．なお，SELinux の動作モードが enforcong [*7] の場合，ログの収集に支障をきたす恐れがるため，permissive [*8] で実験を行った．

手順 1 通常運用中の httpd に関する拒否ログを収集し，audit2allow コマンドによってこれらのアクセスが許されるようなポリシを作成し，SELinux にロードする．この手順は，CI 検出前に許可されていた httpd のアクセスが CI 検出後拒否されることを確認しやすくするために行う．

手順 2 ダミーコマンドを”/usr/local/bin/netcat”に配置する．

手順 3 コマンドインジェクション検出サンプルを使用し，常駐プロセスを起動する．

*7 SELinux による判定を強制する運用モード
*8 SELinux による判定を記録するだけの運用モード

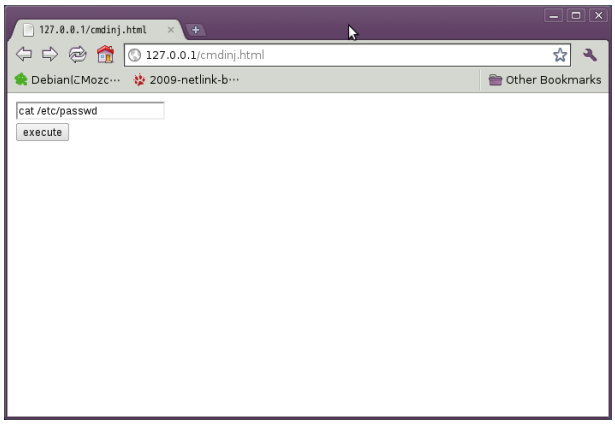

図 6 CI を実行する入力画面：cmdinj.html
Fig. 6 Web Interface for CI: cmdinj.html

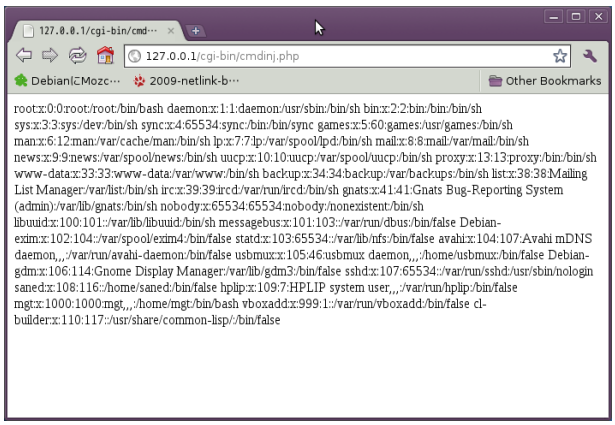

図 7 CI が可能な PHP スクリプト：cmdinj.cgi
Fig. 7 PHP Script for CI: cmdinj.cgi

手順 4 ブラウザから php スクリプトに”cat /etc/passwd”を入力として与えても拒否ログが出力されないことを確認する．これにより httpd によるコマンド実行が許可されていることがわかる．

手順 5 ブラウザから php スクリプトに”netcat –l –p 7777 –e /bin/sh”を入力として与える．これによりダミーの netcat が起動されることで CI が検出され，以降すべての httpd の操作が拒否されることになる．

手順 6 再び 4 と同じ入力を与え，今度は拒否ログが出力されることを確認する．

この実験の結果，CI を通したダミーコマンドの起動によってアクセスベクタキャッシュがクリアされ，それ以降の httpd によるアクセスが拒否されることを確認し，これにより，CI 検出拡張実装が提案方式の上で正しく動作することがわかった．

4.4 考察

評価により，提案方式を用いることで，SELinux のカーネル内実装の変更を伴う諸研究について，一部擬似的な実装ではあるがほぼユーザ空間で実装できることがわかった．一般に，カーネルコードのプログラミングは，資源管理の厳密さやマルチプロセスの扱いの必要性，ライブラリの欠如等の理由から難易度が高く，また，修正のたびにカーネルイメージを再構築する必要もあり，その実装と効果検証に，研究の核以外の多大な労力を要する．従って，提案システムは，アクセス制御モデルやポリシ記述方式を変更するような，従来カーネル内実装の修正を必要とするようなポリシ処理系の改良を簡単化できると考えられる．

一方で，提案方式がユーザ空間で実装可能とする部分は，アクセス主体のセキュリティ情報とアクセス対象のセキュリティ情報・セキュリティクラスからアクセスベクタを求める部分である．従って，どのような操作が行われようとしたかは検出できないし，その結果の可否も検出できないため，SELinux の不要なポリシを減らす研究のような，これらの情報を必要とする仕組みは完全には実装できない．加えて，提案方式では，その構成上，例外的にすべてのアクセスを許可するプロセスが存在するが，通常の SELinux ではすべてのプロセスに対して認可判定を強制することが基本であるため，その差異が完全な検証を困難にするケースがあり得る．結果として，提案方式の上で実装した検証環境では動作したにも関わらず，カーネル空間に実装した場合は動作しない，といった事象が発生する可能性がある．

また，認可判定部分のプロセス間の依存関係や実行の優先順位によりデッドロック等が発生したり，判定プログラムが資源不足等により意図せず終了する可能性が考えられるが，これらについては利用者が留意して使用する必要がある．

5. おわりに

本研究では，カーネル内実装の変更が必要な SELinux のポリシ処理系に関する研究について，実装を簡単化し，効果検証を容易にするため，認可判定処理をユーザ空間で行う方式を提案した．

本稿では，その評価として，SELinux のポリシ処理系に関する先行研究を複数取り上げ，提案方式を使用した場合としない場合について，カーネル内実装の変更項目数を比較した．その結果，これらの研究が完全に同等ではないものの，検証環境としてユーザ空間で再現可能であり，提案方式を使用することで，カーネル空間に実装する場合と比べて容易に実装可能であることがわかった．さらに，提案方式の上で，CI 検出の拡張実装を用意し，新しいアクセス制御モデルを容易に実装できることを示した．

提案方式は，SELinux の主たるカーネル内実装を維持したまま，ユーザ空間に実装したポリシ処理系と連携するように設計されているため，SELinux に用意されているライブラリや専用ツール群，カーネル内実装をほぼそのまま利用可能である．同時に，ポリシ処理系をユーザ空間に実装することで，SELinux のポリシに関する課題解決に向けた試みについて，Linux カーネルの修正や再構成なしに容易に実装し，効果検証できる環境を構成できる．提案方式により，SELinux の欠点を補う改良や機能拡張の研究が容易に行えるようになり，結果として，SELinux がより安全で利用しやすいものになることを期待するものである．

参考文献

[1] Loscocco, P. and Smalley, S.: Integrating Flexible Support for Security Policies into the Linux Operating System, *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, Berkeley, CA, USA, USENIX Association, pp. 29–42 (2001).

[2] Zhai, G., Ma, W., Tian, M., Yang, N., Liu, C. and Yang, H.: Design and Implementation of a Tool for Analyzing SELinux Secure Policy, *Proceedings of the 2Nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, ICIS '09, New York, NY, USA, ACM, pp. 446–451 (online), DOI: 10.1145/1655925.1656007 (2009).

[3] Hicks, B., Rueda, S., St.Clair, L., Jaeger, T. and McDaniel, P.: A Logical Specification and Analysis for SELinux MLS Policy, *ACM Trans. Inf. Syst. Secur.*, Vol. 13, No. 3, pp. 26:1–26:31 (online), DOI: 10.1145/1805874.1805982 (2010).

[4] Sarna-Starosta, B. and Stoller, S. D.: Policy analysis for security-enhanced linux, *Proceedings of the 2004 Workshop on Issues in the Theory of Security (WITS)*, Citeseer, pp. 1–12 (2004).

[5] Jaeger, T., Sailer, R. and Zhang, X.: Analyzing Integrity Protection in the SELinux Example Policy, *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, SSYM'03, Berkeley, CA, USA, USENIX Association, pp. 5–5 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1251353.1251358⟩ (2003).

[6] 山口 拓人, 中村 雄一, 田端 利宏：SELinux の不要なセキュリティポリシの削減手法の提案 (セッション A-2:セキュリティポリシ)，情報処理学会研究報告. CSEC, [コンピュータセキュリティ]， Vol. 2008, No. 21, pp. 37–42（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110006821344/⟩ (2008).

[7] 矢儀 真也，中村 雄一，山内 利宏：SELinux の不要なセキュリティポリシ削減手法の設計と評価 (サービス管理, 運用管理技術, セキュリティ管理, 及び一般)，情報処理学会論文誌コンピューティングシステム（ACS），Vol. 5,

No. 2, pp. 63–73 (2012).

[8] Marouf, S., Phuong, D. M. and Shehab, M.: A Learning-based Approach for SELinux Policy Optimization with Type Mining, *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, CSIIRW '10, New York, NY, USA, ACM, pp. 70:1–70:4 (online), DOI: 10.1145/1852666.1852746 (2010).

[9] 橋本 正樹，金 美羅，辻 秀典，田中 英彦：論理プログラミングを基礎とした認可ポリシ記述言語，情報処理学会論文誌，Vol. 51, No. 9, pp. 1682–1691（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110007970770/⟩ (2010).

[10] 保理江 高志，桝本 圭，宮本 洋輔，原田 季栄，田中 一男：OS カーネルにおける動的アクセス制御，情報処理学会研究報告. CSEC, [コンピュータセキュリティ], Vol. 2003, No. 126, pp. 25–30（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110002664784/⟩ (2003).

[11] Murray, A. P. and Grove, D. A.: PULSE: A Pluggable User-space Linux Security Environment, *Proceedings of the Sixth Australasian Conference on Information Security - Volume 81*, AISC '08, Darlinghurst, Australia, Australia, Australian Computer Society, Inc., pp. 19–25 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1385109.1385115⟩ (2008).

[12] Wright, C., Cowan, C., Smalley, S., Morris, J. and Kroah-Hartman, G.: Linux Security Modules: General Security Support for the Linux Kernel, *Proceedings of the 11th USENIX Security Symposium*, Berkeley, CA, USA, USENIX Association, pp. 17–31 (online), available from ⟨http://dl.acm.org/citation.cfm?id=647253.720287⟩ (2002).