Vol.2018-CSEC-82 No.30
Vol.2018-SPT-29 No.30
2018/7/25

SELinux CILを利用した不要なポリシ削減手法の提案

齋藤 凌也[1]　山内 利宏[1,a)]

概要：SELinuxのセキュリティポリシは設定が難しく，配布されている汎用的なポリシを用いる場合が多い．ここで，配布されているポリシは，個々のシステムに必要ない権限を許可している可能性がある．この問題を解決するため，我々は，実行対象のシステムに合わせて，不要なポリシを自動で削除する手法を提案した．しかし，この手法には，大きく二つの問題が存在する．一つ目は，ポリシのソースファイルが存在しない場合，適用できないという問題である．二つ目は，ポリシ削減の粒度が粗いという問題である．そこで，本稿では，上記二つの問題に対処するため，拡張した手法を提案する．拡張した提案手法では，SELinuxが保持しているSELinux Common Intermediate Languageという中間言語で記述されたファイルに着目し，これを利用することで一つ目の問題に対処する．また，ポリシを削減する際の比較に，アトリビュートを元のタイプに置き換えたアクセスルールを利用することで，二つ目の問題に対処する．本稿では，拡張した手法とその評価結果について報告する．

Ryoya Saito[1]　Toshihiro Yamauchi[1,a)]

1. はじめに

ソフトウェアの脆弱性を利用した攻撃により，システムの改変や機密情報漏洩などの被害が発生している [1]．特に被害が大きくなる可能性の高い攻撃として，ゼロデイ攻撃と権限昇格攻撃がある．ゼロデイ攻撃は，パッチなどの修正手段が提供されていない未知の脆弱性に対する攻撃であり，対処が難しい．また，権限昇格攻撃により，管理者権限が奪取されてしまった場合，攻撃者によって本来禁止されている操作が行われ，被害が大きくなる．

これらの被害を抑制する手段として，Security-Enhanced Linux [2]（以降，SELinux）の利用がある．例えば，Apache Struts 2 の Struts REST プラグインの脆弱性 (CVE-2017-9805) は，SELinuxを有効にしているシステムでは，脆弱性の被害を抑制できるとの報告がある [3]．SELinuxが持つ代表的なアクセス制御機能として，強制アクセス制御 (MAC: Mandatory Access Control) がある．MACとは，アクセス権限の管理者が定めたセキュリティポリシ（以降，ポリシ）のもとで，すべてのファイルやプログラムのアクセス権限が一元的に管理され，所有者が設定を変更できないアクセス制御方式である．SELinuxは，全てのプロセスとリソースにそれぞれドメインとタイプと呼ばれるラベルを付与する．これらのラベルを付与したうえで，ドメインがタイプに対してどのような操作が可能かを定めるアクセスルールを定義する．

しかし，SELinuxのポリシには，ポリシ記述の難しさ，最小特権実現の難しさ，およびポリシのメモリ使用量の多さという3つの問題点がある．SELinuxのポリシはホワイトリスト方式を採用しており，アプリケーションを正常に動作させるためには，多くのアクセスルールを記述する必要がある．また，多くのリソースに対して適切なラベルを付与する必要がある．このため，ポリシの記述は難しい．ポリシは記述の難しさゆえに，多くの場合，コミュニティの開発者によって配布されているポリシがそのまま利用される．ここで，配布されているポリシは汎用的なポリシであり，個々のシステムには必要のないポリシを含んでいる可能性がある．このため，動作しているシステムの最小特権とは差が生じる．ここで，最小特権とは，各プロセスに用途に合った必要最小限のアクセス権限のみを与え，必要以上のアクセス権限を与えないことである．また，不要なポリシがカーネルにロードされるため，メモリの使用量が増加する．

これらの問題を解決するため，文献 [4] では，不要なポリシを自動で削減する手法（以降，従来手法）が実現されている．従来手法は，監査システムが出力するログ（以降，

[1] 岡山大学 大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University
[a)] yamauchi@cs.okayama-u.ac.jp

監査ログ）から変換したポリシと運用中のポリシを比較することで，不要なポリシを発見し，取り除く．しかし，従来手法には 2 つの問題点がある．

1 つ目の問題点は，ポリシのソースファイルが存在しない場合，従来手法を適用できない点である．これは，従来手法にポリシのソースファイルを利用する処理があるためである．2 つ目の問題点は，ポリシ削減の粒度が粗いという点である．これは，アトリビュートを考慮していないためである．ここで，アトリビュートとは，タイプを複数束ねてグループ化したものである．アトリビュートを含むポリシを削除対象としていないため，より細かい粒度でポリシを削除できていない．これは，アトリビュートを含むポリシを削除した場合，複数のタイプに関する権限を一度に削除してしまい，必要なポリシまで削除する可能性があるためである．

そこで，ポリシのソースファイルが存在しない場合においても，不要なポリシを削減する方法を提案する．具体的には，SELinux が保持している SELinux Common Intermediate Language という中間言語で記述されたファイルに着目し，これを利用することで，ソースファイルを利用せずに不要なポリシを削除する．これにより，ポリシをコンパイルし，適用する手間を削減することができる．また，アトリビュートを考慮したポリシ削減の方法を提案する．具体的には，アトリビュートを含むポリシを削減する際に，アトリビュートを元のタイプに置き換え，権限を細分化する．ポリシを削減する際の比較に，置き換えたアクセスルールを利用することで，システムに不要な権限のみを削除する．これにより，システムに必要な権限のみを残し，最小特権に近づけることができる．

2. SELinux

2.1 SELinux のアクセス制御方式

SELinux は，National Security Agency を中心とするコミュニティで開発されている Linux Security Modules である．SELinux の代表的なアクセス制御機能として，MAC がある．MAC は，アクセス権限の管理者が定めたポリシのもとで，全てのファイルやプログラムのアクセス権限が一元的に管理されるアクセス制御方式である．アクセス制御の設定変更は，アクセス権限の管理者のみが行うことができ，リソースの所有者は，アクセス制御の設定を自由に変更できない．SELinux は，Type Enforcement，Role Based Access Control，および Multi Level Security などのアクセス制御機能を実現している [5]．

2.2 SELinux のセキュリティポリシ

2.2.1 Reference Policy

SELinux では，セキュリティポリシと呼ばれる設定ファイルで定義された権限をプロセスに許可することでアクセス制御を行う．Fedora や CentOS では，Reference Policy [6]（以降，refpolicy）というポリシが利用されている．refpolicy は，多くの環境で問題なく動作するように権限が与えられた汎用的なポリシである．また，ポリシがモジュール化されており，ポリシの運用中でも，モジュール単位でポリシの追加や削除が可能である．refpolicy はカーネルポリシ言語で記述されており，以下の 3 つのファイルから構成される．

(1) fc（file context）ファイル
　プロセスやシステム資源のパス名とラベルの対応付けを記述する．
(2) if（interface）ファイル
　te ファイルで使用するマクロを記述する．
(3) te（type enforcement）ファイル
　アクセス権限の付与を記述する．

TE は記述したアクセルルールのみを許可するホワイトリスト方式を採用している．te ファイルに記述されるアクセスルールは，マクロを使用しない場合，以下の 5 つの要素で構成される．カーネルポリシ言語におけるアクセスルールの記述を**図 1** に示す．

(1) ポリシルール（`rule_name : allow, auditallow, dontaudit, neverallow`）
(2) ドメイン（`domain`）
(3) タイプ（`type`）
(4) オブジェクトクラス（`class`）
(5) アクセスベクタパーミッション（`av`）

ポリシルールの `allow` はアクセスを許可することを意味し，`neverallow` はアクセスを許可しないことを意味する．`auditallow` は監査ログのうち，アクセスを許可した際のログ（以降，許可ログ）を出力させることを意味する．また，`dontaudit` は監査ログのうち，アクセスを拒否した際のログ（以降，拒否ログ）を出力しないようにすることを意味する．ドメインはプロセスのラベルであり，タイプは操作対象となるリソースのラベルである．オブジェクトクラスは，ファイルやディレクトリのようにオブジェクトの種類を分類するものである．アクセスベクタパーミッションは，読み取り権限や書き込み権限のようなアクセスパーミッションであり，オブジェクトごとに定義されている．

refpolicy を利用する方法として，公開されているソースファイルをダウンロードする方法と，提供されているパッケージを利用する方法がある．公開されているソースファイルをダウンロードする方法では，ソースファイルをダウンロードした後，コンパイルし，システムに適用する必要がある．ポリシをシステムに適用する流れを**図 2** に示す．

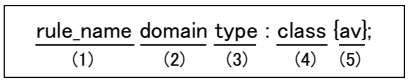

図 1　カーネルポリシ言語におけるアクセスルールの記述

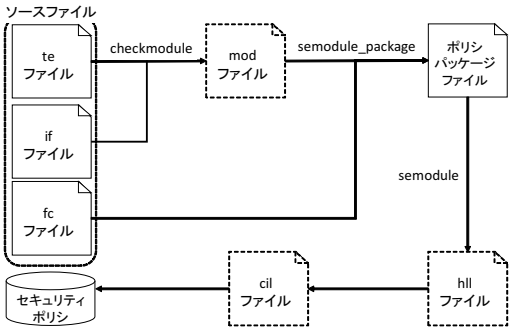

図 2 ポリシコンパイルの流れ（SELinux のバージョン 2.4 以降）

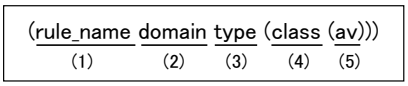

図 3 SELinux CIL におけるアクセスルールの記述

ソースファイルは，`checkmodule` コマンドと `semodule_package` コマンドにより，中間ファイルを経て，ロード可能なモジュールファイル（ポリシパッケージファイル）にコンパイルされる．このポリシパッケージファイルは高水準言語 (hll) ファイルと呼ばれ，中間言語 (cil) ファイルに変換される．最後に，全ての cil ファイルをまとめて，バイナリのポリシにコンパイルする．ポリシパッケージファイルからバイナリのポリシに変換されるまでの流れは，`semodule` コマンドにより処理される．なお，SELinux のバージョン 2.4 より前は，hll ファイル，cil ファイルが導入されておらず，ポリシパッケージファイルからバイナリのポリシにコンパイルされる．

提供されているパッケージを利用する方法では，バイナリのポリシ，hll ファイル，cil ファイル，およびその他の設定ファイルがパッケージとして提供されており，ポリシを切り替えるだけで利用できる．

2.2.2 SELinux Common Intermediate Language

SELinux Common Intermediate Language（以降，SELinux CIL）とは，高水準言語とバイナリのポリシとの中間言語となるように設計された言語 [7] であり，SELinux のバージョン 2.4（2015 年 2 月リリース）以降から，Fedora では，Fedora 23（2015 年 10 月リリース）から取り込まれている [8]．アクセスルールとして記述されるポリシは，2.2.1 項で述べた 5 つの要素で構成される．SELinux CIL におけるアクセスルールの記述を**図 3** に示す．SELinux CIL は解析やポリシの生成を簡単にするために設計されている．このため，図 3 のように，SELinux CIL の文法には解析しやすいリスト構造（S 式）が使用されている．

2.3 ポリシの問題点

2.3.1 ポリシ記述の難しさ

SELinux のポリシ記述を難しくする要因として，アクセスルールの総数とアクセスルールの記述がある [9]．

アクセスルールの総数

SELinux のセキュリティポリシはホワイトリスト方式を採用しているため，アプリケーションを正常に動作させるためには多くのアクセスルールが必要となる．ここで，パーミッションの種類は 700 を超えるため，アクセスルールの数が増大する．実際に，Fedora 9 において，デフォルトで利用されているポリシ内のアクセスルールの数は 150,000 を超える．

アクセスルールの記述

(1) パーミッションの設定

SELinux のパーミッションは，システムコールの観点から設計されている．このため，Linux カーネルの知識が必要になる．さらに，700 を超える種類のパーミッションがパーミッションの設定をより難しくする．

(2) ラベルの設定

システム内全てのファイルとポートに対してラベルを付与する必要がある．ここで，標準の Linux システムには 10,000 を超えるファイルが存在する．このため，ラベルの設定は非常に難しい．

ポリシ記述の難しさを解決するため，SELinux のポリシ設定ツールとして，SEEdit [10] などの様々なツールが開発されている．しかし，これらのツールを利用したとしても計算機システムの知識が必要であり，設定工数が多いため，一からポリシを記述することは簡単ではない．

2.3.2 最小特権実現の難しさ

最小特権とは，各プロセスに用途に合った必要最小限のアクセス権限のみを与え，必要以上のアクセス権限を与えないことである．SELinux のポリシはホワイトリスト方式を採用しているため，ポリシに記述されている操作以外は全て禁止される．このため，ブラックリスト方式に比べ，安全性が高い．一方で，誤って必要以上の権限を与えることで，安全性を低下させる可能性がある．必要以上の権限を与えてしまう原因として以下の 2 つがある．

(1) 配布されているポリシ（refpolicy）の利用

ポリシ記述の難しさから，自分でポリシを作成することなく，コミュニティの開発者が配布しているポリシを利用する場合が多い．配布されているポリシは汎用的なポリシであり，利用していないデーモンやアプリケーションに関するポリシを含んでいるため，個々のシステムには必要のない権限を許可している可能性がある．また，利用しているアプリケーションでも，多くの環境で問題なく動作するように多くの権限が与えられている．このため，動作しているシステムの最小特権とは差が生じる．

(2) 拒否ログからポリシを自動生成

拒否ログからポリシを生成するツールとして

audit2allow コマンドがある．このコマンドは，監査ログを解析し，アクセス拒否された操作について，アクセス許可を追加するためのポリシを自動生成する．ここで，ポリシを生成する際，本来許可してはならない権限まで，システムの管理者が誤って許可する可能性がある [11]．必要以上の権限を許可した場合，ホワイトリスト方式では，これを見つけることは困難である．

2.3.3 ポリシのメモリ使用量の多さ

IoT 機器に利用されている OS の約 86%が Linux であり [12]，提供する機能が限られていることから，SELinux を適用し，最小限のポリシでセキュリティを強固にできる．一方で，Fedora 27 においてデフォルトで利用されている refpolicy のメモリ使用量は約 3.7MB である．IoT 機器のようにメモリサイズが限られている場合には，必要最小限のポリシを作成し，メモリ使用量を削減する必要がある．

3. SELinux CIL を利用した不要なポリシ削減手法

3.1 従来手法の問題点

2.3 節で述べた問題点を解決するため，文献 [4] では，不要なポリシを自動で発見し，削除する手法を実現している．従来手法では，汎用的なポリシである refpolicy に含まれている不要なポリシを発見し，削除する．不要なポリシを発見する方法として，SELinux が出力する監査ログを利用する．現在運用中のポリシと許可ログから作成したポリシを比較することで，不要なポリシを発見し，削除する（以降，ポリシ削減機能）．また，誤って必要なポリシを削除した場合のために，誤って削除したポリシを発見し，復元する（以降，ポリシ復元機能）．ポリシ復元機能も，監査ログを利用して実現される．SELinux に許可ログを出力させるために，従来手法は auditallow 文を利用している．一方で，従来手法には，以下の問題点が存在する．

（問題点 1） ポリシのソースファイルがない場合，従来手法を適用できない

従来手法は，ポリシのソースファイルを利用する処理がある．このため，文献 [4] では，ポリシのソースファイルをダウンロードした後，コンパイルし，適用したシステムで評価を行っていた．一方で，最新の Fedora や CentOS において，デフォルトで利用されているポリシはコンパイル済みのバイナリ形式であり，ポリシのソースファイルが含まれていない．また，提供されているパッケージを利用する場合においても，同様にポリシのソースファイルが含まれていない．このため，デフォルトのポリシを利用しているシステムや提供されているパッケージを利用しているシステムでは，従来手法を適用できない．

（問題点 2） ポリシ削減の粒度が粗い

```
(typeattribute httpd_script_exec_type)
(typeattributeset httpd_script_exec_type (httpd_sys_script_exec_t httpd_user_script_exec_t))
        :
(allow httpd_t httpd_script_exec_type (file (ioctl read getattr lock open)))
        :
```

図 4 アトリビュートの宣言例

従来手法は，アトリビュートを含む allow 文をポリシ削減の対象としていない．ここで，アトリビュートとは，タイプを複数束ねてグループ化したものである．例えば，SELinux CIL では図 4 のように宣言し，利用される．図 4 の例では，attribute 宣言で httpd_script_exec_type というアトリビュートを宣言し，typeattributeset 宣言で，httpd_script_exec_t と httpd_user_script_exec_t という 2 つのタイプをグループ化している．図 4 の allow 文では 2 つのタイプに対してアクセスを許可していることになる．従来手法では，アトリビュートを含む allow 文を削除せず，必要なポリシとして残している．このため，従来手法のポリシ削減は粒度が粗い．

3.2 提案手法

3.2.1 課題

3.1 節で述べた 2 つの問題点を解決するために，従来手法を拡張した不要なポリシ削減手法（以降，提案手法）を提案する．提案手法を実現するための課題を以下に示す．

（課題 1） ポリシのソースファイルを利用せずにログを出力させる方法

従来手法では，ポリシのソースファイルに auditallow 文を追加することで，ログを出力させていた．しかし，ポリシのソースファイルが存在しない場合は，別の方法を検討する必要がある．

（課題 2） ポリシのソースファイルで定義されているアクセスルールを取得する方法

運用中のポリシから不要なポリシを発見するためには，ログから生成されたポリシと運用中のポリシを比較する必要がある．このため，何らかの方法でポリシのソースファイルで定義されているアクセスルールを取得する必要がある．

（課題 3） アトリビュートを含むポリシの削減方法

アトリビュートを含むポリシを削除した場合，複数のタイプの権限を削除してしまい，システムに必要な権限のみを残すことができない．このため，アトリビュートを含むポリシを削除する際，システムに不要な権限のみを削除する方法を検討する必要がある．

（課題 1）と（課題 2）は，（問題点 1）に，（課題 3）は，（問題点 2）にそれぞれ対応している．

3.2.2 対処

各課題への対処を以下に示す．

（対処 1） ポリシのソースファイルを利用せずにログを

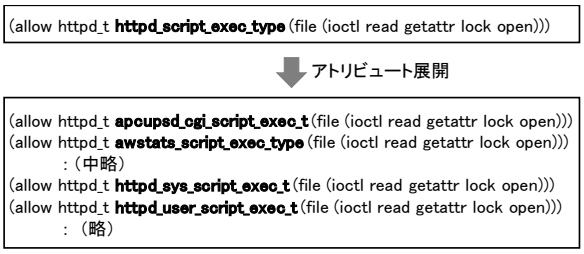

図 5　アトリビュート展開の例

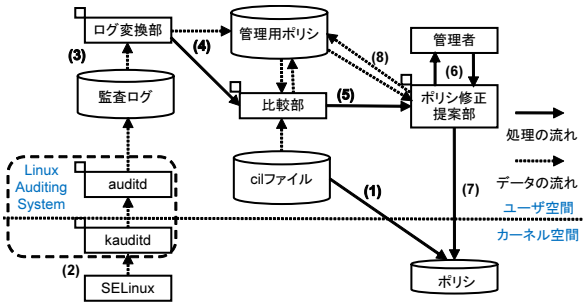

図 6　提案手法におけるポリシ削減機能の処理流れ

出力させる方法

cil ファイルを利用する．図 2 のように，ポリシパッケージファイルは，cil ファイルに変換される．SELinux は，cil ファイルを保持している．このため，cil ファイルに `auditallow` 文を追加し，システムに適用することで，ログを出力させることができる．

（対処 2）　ポリシのソースファイルで定義されているアクセスルールを取得する方法

(対処 1) と同様に cil ファイルを利用する．cil ファイルを利用することで，ポリシのソースファイルで定義されているアクセスルールを取得できる．

（対処 3）　アトリビュートを含むポリシの削減方法

アトリビュートを含むポリシを削減する際に，アトリビュートを元のタイプに置き換え，権限を細分化する（以降，アトリビュート展開）．アトリビュート展開の例を**図 5** に示す．比較部において，ログから作成したポリシと定義されたアクセスルールを比較する際に，アトリビュート展開後のポリシを比較することにより，システムに必要な権限のみを残すことができる．

3.3　処理流れ

提案手法は，ログ収集とポリシ削減期間，テスト運用期間，および実運用期間の 3 つに分類される．ログ収集とポリシ削減期間では，ポリシ削減機能を利用して，不要なポリシを削除する．その後，テスト運用期間でポリシ復元機能を利用して，誤って削除してしまったポリシを復元する．提案手法におけるポリシ削減機能の処理流れを**図 6** に示し，以下で説明する．

(1) cil ファイルに `auditallow` 文を追加し，システムに適用
(2) SELinux がログを出力
(3) (2) を一定期間繰り返し，ログを収集した後，ログ変換部を起動
(4) 収集したログをログ変換部がポリシに変換し，管理用ポリシに保存
(5) ログから作成したポリシと cil ファイルを比較部が比較し，差分のポリシを作成した後，ポリシ修正提案部を起動
(6) ポリシ修正提案部が差分のポリシの内容を通知し，ポリシの修正を提案
(7) 管理者がポリシの修正を許可した場合，ポリシ修正提案部が cil ファイルを修正し，システムに反映
(8) ポリシ修正提案部が修正したポリシを管理用ポリシに保存

また，提案手法におけるポリシ復元機能の処理流れを以下で説明する．

(1) SELinux が拒否ログを出力
(2) ログ変換部が audit dispatcher daemon から拒否ログを受信し，ポリシに変換後，比較部を起動
(3) 拒否ログから作成したポリシと管理用ポリシを比較部が比較し，以前に削除したポリシと一致するか否かを調査
(4) 一致した場合，比較部がポリシ修正提案部を起動し，ポリシ修正提案部がポリシの復元を提案
(5) 管理者がポリシの修正を許可した場合，ポリシ修正提案部が cil ファイルを修正し，システムに反映
(6) ポリシ修正提案部が修正したポリシを管理用ポリシに保存

3.4　期待される効果

提案手法を適用することで，以下の効果が期待される．

(1) ポリシをパッケージで利用する場合でも削減手法を適用可能

refpolicy を利用する方法として，ポリシのソースファイルをダウンロードする方法と提供されているパッケージをダウンロードする方法がある．後者の方法は，ソースファイルが含まれていないため，従来手法を適用できなかった．しかし，SELinux CIL 形式のポリシを利用する提案手法により，両者の方法に対して不要なポリシを削減可能となる．これにより，ポリシをコンパイルし，適用する手間を削減することができる．

(2) より粒度の細かいポリシ削減が可能

アトリビュート展開後のポリシを比較部で利用することにより，システムに不要な権限のみを削除できるため，より粒度の細かいポリシ削減が可能である．これにより，最小特権に近づけることができる．

4. 評価

4.1 評価内容と評価環境

提案手法の有用性を明らかにするために以下の評価を行った．

（評価 1） アトリビュート展開による権限の細分化
提案手法がアトリビュートを含むポリシを削減する際に，システムに必要な権限のみを残すことができているかを示す．

（評価 2） ポリシの削減量
提案手法を適用することにより，allow 文の数とポリシのサイズをどの程度削減できるかを示す．

（評価 3） 提案手法適用によるアプリケーションの性能への影響
提案手法を適用することで発生するオーバヘッドを計測し，アプリケーションの性能への影響を示す．

（評価 4） ポリシコンパイルの手間削減
従来手法と提案手法を導入するまでの手順を比較することで，ポリシをコンパイルする手間がどの程度削減できるかを示す．

評価環境は，カーネルは，Linux 4.13.16-302.fc27.x86_64 (Fedora 27)，CPU は Intel(R) Core(TM) i5-6500 3.20GHz，メモリは 4GB，ポリシのバージョンは selinux-policy-targeted-3.13.1-283.17.fc27 である．

4.2 アトリビュート展開による権限の細分化

apache モジュールで定義されている allow 文の例を以下に示す．

```
(allow httpd_t file_type (dir (getattr
search open)))
```

上記の例は，アトリビュート **file_type** を含む allow 文である．file_type は，2,944 のタイプを束ねているアトリビュートである．このため，上記の allow 文では，2,944 のタイプに対してアクセスを許可している．提案手法適用後，上記の allow 文は**図 7** に示す allow 文に置き換えられた．

図 7 では，25 タイプに対してアクセスを許可している．また，httpd_config_t には 3 つの権限（getattr，search，open）を与えている．また，httpd_log_t，httpd_sys_contents_t，および var_t には 2 つの権限（getattr, search），その他タイプには 1 つの権限（search）をそれぞれ残し，不要な権限を削除していることがわかる．このことから，アトリビュートを含む allow 文を削減する際，各タイプに必要な権限を残し，不要な権限を削除していることがわかる．なお，上記以外のアトリビュートを含む allow 文の削減においても，同様の結果が得られた．

以上より，提案手法は，アトリビュートを含む allow 文を削減する際，システムに必要な権限のみを残し，最小特権に近づけることができるといえる．

```
(allow httpd_t httpd_config_t (dir (open getattr search)))
(allow httpd_t httpd_log_t (dir (getattr search)))
(allow httpd_t httpd_modules_t (dir (search)))
(allow httpd_t httpd_sys_content_t (dir (getattr search)))
(allow httpd_t httpd_var_run_t (dir (search)))
(allow httpd_t bin_t (dir (search)))
(allow httpd_t device_t (dir (search)))
(allow httpd_t etc_t (dir (search)))
(allow httpd_t home_root_t (dir (search)))
(allow httpd_t proc_t (dir (search)))
(allow httpd_t root_t (dir (search)))
(allow httpd_t sysctl_t (dir (search)))
(allow httpd_t sysfs_t (dir (search)))
(allow httpd_t tmp_t (dir (search)))
(allow httpd_t usr_t (dir (search)))
(allow httpd_t var_lib_t (dir (search)))
(allow httpd_t var_run_t (dir (search)))
(allow httpd_t var_t (dir (getattr search)))
(allow httpd_t chronyd_var_lib_t (dir (search)))
(allow httpd_t colord_var_lib_t (dir (search)))
(allow httpd_t geoclue_var_lib_t (dir (search)))
(allow httpd_t glusterd_var_run_t (dir (search)))
(allow httpd_t init_var_run_t (dir (search)))
(allow httpd_t lib_t (dir (search)))
(allow httpd_t var_log_t (dir (search)))
```

図 7 置き換えられた allow 文

4.3 ポリシ削減量

allow 文の数とポリシのサイズを評価した．allow 文の数は，最小特権に近づけているかを示すための評価であり，ポリシのサイズは，メモリ使用量の削減についての評価である．HTTP サーバ，SFTP サーバ，SMB サーバ，および DNS サーバが動作している計算機に対して提案手法を適用し，ポリシの削減量の評価を行った．全部で 413 個ある refpolicy のモジュールの内，apache モジュール，ssh モジュール，samba モジュール，および bind モジュールに関するポリシ削減を行う．なお，サーバの利用人数は 8 人とし，ログを収集する期間は 2 日間である．本評価の手順を以下に示す．

(1) 上記 4 つのモジュールに auditallow 文を適用し，許可ログの収集を開始
(2) 計算機を再起動
(3) 2 日間許可ログを収集
(4) 不要なポリシを削除

allow 文の削減数とポリシサイズの削減量をそれぞれ**表 1** と**表 2** に示す．

表 1 の合計の削減数から，本環境では，4 つのモジュールで定義されている allow 文の 9 割以上は，実際には利用されておらず，不要な権限であることがわかる．このことから，提案手法は，モジュールに含まれる不要なポリシを削減でき，最小特権に近づけることができるといえる．また，表 2 から，ポリシのサイズは 0.9%削減できたことがわかる．ポリシサイズの削減量の割合が 0.9%と非常に小さい原因は，削減対象としたモジュールの数が少ないためであると推察している．全てのモジュールに提案手法を適用することで，ポリシサイズの削減量は大きくなる可能性がある．

表 1　allow 文の削減数

モジュール名	デフォルトの allow 文の数	アトリビュート展開後の allow 文の数 (N1)	ポリシ削減後の allow 文の数 (N2)	削減数 (%) (N1-N2)
apache	1,242	85,695	86	85,609 (99.9%)
bind	211	4,649	59	4,590 (98.7%)
samba	1,089	34,182	97	34,085 (99.7%)
ssh	714	14,562	91	14,471 (99.4%)
合計	3,256	139,088	333	138,755 (99.8%)

表 2　ポリシサイズの削減量

	デフォルト	削減後	削減量 (%)
ポリシのサイズ (B)	3,880,828	3,844,479	0.9

表 3　全てのリクエスト完了に要した処理時間 (単位：s)

許可ログ非出力時	許可ログ出力時	オーバヘッド
14.349	37.961	23.612 (164.56%)

4.4　提案手法適用によるアプリケーションの性能への影響

提案手法適用によるアプリケーションの性能への影響を評価するために，許可ログを出力している期間と許可ログを出力していない期間における Web サーバの性能を測定した．性能を測定するために ApacheBench を利用し，全てのリクエスト完了に要した処理時間を比較することで，アプリケーションの性能への影響を考察する．10KB のファイルに対し，合計リクエスト 100,000 で，同時接続数が 10 の場合を 5 回測定し，実行時間の平均を算出した．評価に用いた Web サーバは，Apache 2.4.29 である．本評価では，ApacheBench 2.3 を用いた．クライアント側の環境は，CPU は Intel(R) Core(TM) i5-6500 3.20GHz，メモリは 4GB，カーネルは Linux 4.4.0 である．Apache に対応するモジュールである apache モジュールに提案手法を適用し，許可ログを出力させた．測定結果を**表 3** に示す．許可ログを出力させることによるオーバヘッドは 23.612 秒 (164.56%) となった．ここで，このオーバヘッドは，不要なポリシを削減するために許可ログを収集している期間で生じるものである．このため，計算機の性能は落ちるものの，サービスの継続が不可能となるほど処理時間が増加しないことから，許容できるオーバヘッドであるといえる．

4.5　ポリシコンパイルの手間削減

従来手法と提案手法を導入するまでの手順を以下に示す．この評価では，文献 [5] で説明されている Fedora の例で説明する．

従来手法の導入手順

(1) ポリシのソースファイル (refpolicy) をダウンロード
(2) ダウンロードした refpolicy に Fedora 用のパッチを適用
(3) 設定ファイル等を適切なディレクトリに配置
(4) ポリシの設定ファイル (build.conf) を記述
(5) ポリシをコンパイル
(6) 利用するポリシを変更 (/etc/selinux/config ファイルに利用するポリシ名を記述)

(3) で，適切なディレクトリに配置できていない場合，SELinux によるアクセス拒否が発生し，OS が動作を停止する可能性がある．(4) では，ポリシ名や，ポリシをモジュール化するかどうかなどの設定を記述する．このため，(3)，(4) を行うためには，SELinux のポリシに関する知識が必要となる．

提案手法の導入手順

(1) Fedora 用のポリシパッケージをダウンロード
(2) 利用するポリシを変更

提案手法では，ポリシのソースファイルが存在しない場合においても，SELinux が保持している SELinux CIL 形式のポリシを利用することで，ポリシ削減を行うことが可能である．このため，従来手法のように，ポリシをソースファイルからコンパイルする必要がない．また，SELinux のポリシに関する知識は必要ない．以上から，提案手法は，ポリシ削減の手間を削減できるといえる．

5.　関連研究

ポリシ作成の工程数を減らす研究として，文献 [13] がある．文献 [13] は，全てのアクセスを許可するルールをポリシに記述した後，ユーザによって指定された箇所のみのアクセス制限を GUI で行う．これにより，ポリシ作成に必要な前提知識の量と作業量が少なく済むことが期待される．一方で，ユーザの設定し忘れ等により，アクセス制限が行われていないリソースが存在した場合，ポリシの安全性が低下する欠点がある．また，ユーザがアクセス制限を指定する必要があるため，設定を行うユーザがシステムに詳しい必要がある．一方で，提案手法では，既存のポリシから自動で不要なポリシを発見し，取り除く．このため，システムの管理者に必要な知識が少なくて済む．

収集したログからポリシを作成する研究として，文献 [14] と文献 [15] がある．文献 [14] は，一定期間システムを稼働させ，プログラムの実行履歴とアクセス要求の情報を収集することによりポリシを作成する．履歴を収集している間にアクセス拒否が発生した場合は無視され，アクセス拒否が発生しないように必要なアクセスルールをポリシに追

加する．一方，提案手法は，不要なポリシを削減することを目的としており，許可ログを収集している期間にポリシに記述されていない操作が行われた場合はアクセスを拒否する．このため，元々のポリシに記述されていないアクセスルールが追加されることはない．

文献 [15] では，大規模な監査ログとポリシを解析し，SEAndroid のポリシを洗練化するプラットフォームを提案している．このプラットフォームでは，ドメインやタイプなどの 6 つの項目をアクセスパターンとして定義する．まず，既存のポリシから良性と悪性のアクセスパターンをそれぞれ抽出し，これを正解データとする．次に，監査ログからアクセスパターンを抽出し，半教師あり学習により，良性と悪性のアクセスパターンにそれぞれ分類する．この分類結果を正解データにマージし，新たな正解データが生成されなくなるまで繰り返す．最後に，分類結果のアクセスパターンからアクセスルールやラベルを生成し，ポリシにマージする．文献 [15] は，半教師あり学習を利用して，主に拒否ログからポリシを作成するのに対し，提案手法は，監査ログを利用して，不要なポリシを発見し，削除する．

6. おわりに

従来手法には，ポリシのソースファイルが存在しない場合にポリシを削減できない点と，ポリシ削減の粒度が粗い点という 2 つの問題点があることを述べた．これらの問題を解決するために，従来手法の拡張を提案し，その方式と評価結果について述べた．提案手法は，SELinux が保持している SELinux CIL という中間言語で記述されたファイルに着目し，これを利用することでポリシのソースファイルが存在しない場合においても不要なポリシを削減できる．これにより，ポリシをコンパイルし，適用する手間を削減することができる．また，アトリビュートを含むポリシを削減する際に，アトリビュートを元のタイプに置き換え，権限を細分化する．ポリシを削減する際の比較に，置き換えたアクセスルールを利用することで，システムに不要な権限のみを削除する．これにより，最小特権に近づけることができる．

ポリシの削減量の評価として，HTTP サーバ，SFTP サーバ，SMB サーバ，DNS サーバが動作している計算機に対して提案手法を適用し，ポリシの削減を行った．評価結果より，評価環境では，4 つのモジュールで定義されている `allow` 文の 9 割以上が実際には利用されておらず，不要な権限であることがわかった．また，ポリシのサイズを 0.9%削減することができた．さらに，アトリビュート展開によって，アトリビュートを含む `allow` 文を削減する際，システムに不要な権限のみを削除できることを示した．性能評価では，提案手法によるオーバヘッドが約 165%であり，許可ログを収集している期間のみに発生する一時的なものであることを示した．ポリシコンパイルの手間削減の評価では，従来手法に比べ，提案手法はポリシに関する知識は必要とせず，ポリシコンパイルの手間を削減できることを示した．

残された課題として，全てのモジュールに対して提案手法を適用した際のポリシ削減量の評価がある．

参考文献

[1] 独立行政法人情報処理推進機構：脆弱性対策情報データベース JVN iPedia のに関する活動報告レポート [2017 年第 4 四半期（10 月～12 月）], 入手先 ⟨https://www.ipa.go.jp/files/000063694.pdf⟩（参照 2018-01-29）．
[2] Security-Enhanced Linux，入手先 ⟨https://www.nsa.gov/what-we-do/research/selinux/⟩（参照 2017-12-26）．
[3] Kazuki Omo：S2-052: CVE-2017-9805(Struts2) PoC with SELinux，OSS セキュリティ技術の会（Secure OSS SIG），入手先 ⟨http://www.secureoss.jp/post/omok-selinux-struts2-20170911/⟩（参照 2018-06-13）．
[4] 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の自動化手法の提案，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）．
[5] Richard Haines: The SELinux Notebook(4th Edition), available from ⟨http://freecomputerbooks.com/books/The_SELinux_Notebook-4th_Edition.pdf⟩ (accessed 2017-04-14).
[6] TresysTechnology: SELinux Reference Policy, GitHub, available from ⟨https://github.com/TresysTechnology/refpolicy⟩ (accessed 2017-01-12).
[7] MacMillan, K., Case, C., Brindle, J. and Sellers, C.: SELinux Common Intermediate Language Motivation and Design, GitHub, available from ⟨https://github.com/SELinuxProject/cil/wiki⟩ (accessed 2017-06-07).
[8] Moore, P.: The State of SELinux, Linux Security Summit 2015, available from ⟨http://kernsec.org/files/lss2015/lss-state_of_selinux-pmoore-082015-r1.pdf⟩ (accessed 2017-12-13).
[9] Nakamura, Y., Sameshima, Y. and Yamauchi, T.: SELinux Security Policy Configuration System with Higher Level Language, Journal of Information Processing Vol.18, pp.201–212 (2010).
[10] Nakamura, Y., Sameshima Y. and Tabata, T.: SEEdit: SELinux Security Policy Configuration System with Higher Level Language: Proc. 23rd Large Installation System Administration Conference (LISA'09), pp.107–117 (2009).
[11] Bill McCarty. 田口裕也（監訳），根津研介（監訳），林秀幸（訳）：SELINUX システム管理 セキュア OS の基礎と運用，p.107， オライリー・ジャパン（2005）．
[12] Costin, A., Zaddach, J., Francillon, A. and Balzarotti, D.: A Large-Scale Analysis of the Security of Embedded Firmwares, Proc. 23rd USENIX Security Symposium, pp.95–110 (2014).
[13] 桝本圭，村田裕之：SELinux のポリシ作成時間を短縮する一考察，Japan Linux Conference 抄録集，Vol.1（2007）．
[14] 原田季栄，半田哲夫，橋本正樹，田中英彦：アプリケーションの実行状況に基づく強制アクセス制御方式，情報処理学会論文誌，vol.53，no.9，pp.2130–2147（2012）．
[15] Wang, R., Enck, W., Reeves, D., et al.: EASEAndroid: Automatic Policy Analysis and Refinement for Security Enhanced Android via Large-Scale Semi-Supervised Learning, Proc. 24th USENIX Security Symposium, pp.351–366 (2015).