Computer Security Symposium 2018
22 - 25 October 2018

ログ出力の抑制によるSELinuxの不要なポリシ削減手法

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，設定が難しいという問題がある．ポリシは設定の難しさから，配布されているポリシを利用する場合が多い．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．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と，SELinux が出力する監査ログを利用して不要なポリシを発見し，削除する．ここで，提案手法には，同一ポリシにより許可されたアクセスのログが出力され続ける点，すべてのポリシのモジュールを対象にして提案手法を適用できない点，および提案手法適用時のオーバヘッドが大きい点という 3 つの問題点が存在する．我々は，上記 3 つの問題点に対処するため，提案手法を拡張した．拡張した提案手法では，一度ポリシの形式に変換されたアクセスルールに関する `auditallow` 文をポリシから削除する．これにより，ログの出力が抑えられ，3 つの問題点に対処できる．本稿では，提案手法とその評価結果について報告する．

キーワード：SELinux，セキュリティポリシ，SELinux CIL，強制アクセス制御，最小特権

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

1. はじめに

ソフトウェアの脆弱性によって，情報漏洩や改ざんなどの被害が発生している [1]．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．このアクセス制御方式により，SELinux は高いセキュリティを実現している．

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．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，利用しないデーモンやアプリケーションに関するポリシを含む汎用的なポリシであるため，個々のシステムにはアクセスを許可する必要のないポリシ（以降，不要なポリシ）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] において，SELinux CIL を利用した不要なポリシ削減手法（以降，文献 [4] の手法）を提案した．文献 [4] の手法は，SELinux が保持している SELinux Common Intermediate Language（以降，SELinux CIL）という中間言語で記述されたファイルと SELinux がアクセスを許可した際に出力するログ（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

許可ログ）から変換した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．また，タイプを複数束ねてグループ化したアトリビュートを含むポリシを削減する際に，元のタイプに置き換えて権限を細分化し，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．文献 [4] の手法により，従来手法 [5] の問題点であるポリシのソースファイルが存在しない場合に適用できないという問題とポリシ削減の粒度が粗いという問題を解決した．

しかし，文献 [4] の手法には，次の 3 つの問題点が存在する．

1 つ目の問題点は，ログを収集する期間において，同一ポリシにより許可されたアクセスの監査ログが出力され続ける点である．例えば，Apache において，`httpd_sys_content_t` というタイプが付与されたファイルに 10 回アクセスした場合，`httpd_sys_content_t` に対して `read` を行ったという許可ログが 10 回出力される．このため，収集する監査ログのファイルサイズが巨大になる．これらの許可ログから変換されるポリシは 1 種類であるため，2 回目以降の許可ログの出力は不要である．

2 つ目の問題点は，すべてのポリシのモジュールに対して提案手法を適用できない点である．これは，内容の重複するログが多く出力されるため，すべてのモジュールに対して提案手法を適用し，ログを収集した場合，収集する監査ログのファイルサイズが膨大になるためである．

3 つ目の問題点は，提案手法適用時のオーバヘッドが大きい点である．文献 [4] で行った評価では，apache モジュールに対して提案手法適用した際，ログ収集とポリシ削減期間において，約 164%のオーバヘッドが発生した．これは，出力されるログが膨大であることが原因であると考えられる．

上記の 3 つの問題点を解決するため，本稿では，監査ログの出力を抑えるように提案手法を拡張した．提案手法は，許可ログを出力するために，`auditallow` 文をポリシに追加している．提案手法では，一度ポリシの形式に変換されたアクセスルールに関する `auditallow` 文をポリシから削除する．これにより，削除後は同一ポリシにより許可されたアクセスの監査ログが出力されなくなるため，ログの出力を抑えることが可能である．

本稿では，拡張した提案手法とその評価結果について報告する．

本研究の貢献は，以下に示す通りである．

(1) 最小特権に近づけられること
SELinux のポリシに関する詳しい知識がなくとも，提案手法を利用することで，不要なポリシを削除することにより，削除不可能なモジュールを除いたモジュール（評価では 388 個）に対して，必要以上のアクセス権限を削除することができる．これにより，最小特権に近づけることができる．

(2) ログの出力を抑制できること
一度ポリシの形式に変換されたアクセスルールに関する `auditallow` 文をポリシから削除することで，ログの出力を抑えることが可能である．これにより，提案手法適用によって発生するオーバヘッドが徐々に抑えられていく．

(3) 利用されていないモジュールを発見し，削除できること
ログの出力を抑制したことで，すべてのモジュールに対して，提案手法を適用可能である．利用されていないモジュールを発見し，削除することで，ポリシのサイズを削減することが可能である．

2. SELinux と不要なポリシ削減手法

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 カーネルのセキュリティ拡張機能である．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，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により，高いセキュリティを実現している [6].

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 [7]（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．

2.2.2 SELinux CIL

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 [8] であり，Fedora では，Fedora 23（2015 年 10 月リリース）から取り込まれている [9]．アクセスルールとして記述されるポリシは，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．SELinux CIL 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`，

```
(rule_name domain type (class (av)))
    (1)      (2)   (3)   (4)   (5)
```

図 1 SELinux CIL におけるアクセスルールの記述

```
(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 2 アトリビュートの宣言例

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．タイプは，複数束ねてグループ化することが可能であり，束ねたものをアトリビュートという．アトリビュートの例を**図 2** に示す．図 2 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．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．

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 [10].

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

システム内すべ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 [11] 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すべて禁止される．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 [12]．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 [13]，提供する機能が限られていることから，SELinux

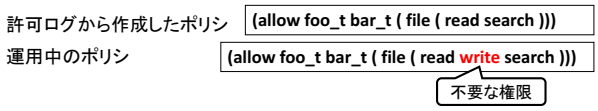

図 3　不要なポリシの発見

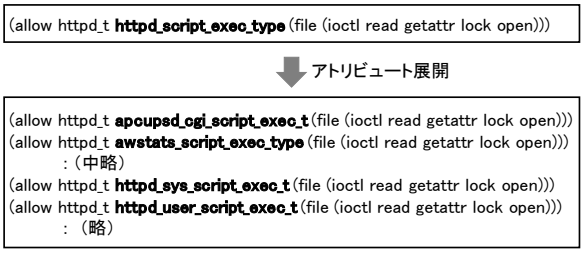

図 4　アトリビュート展開の例

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．

2.4　文献 [4] のポリシ削減手法

2.3 節で述べた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文献 [4] で提案した．文献 [4] の手法では，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と SELinux が出力する監査ログを利用して不要なポリシを発見し，削除する．提案手法では，SELinux が出力する許可ログを一定期間収集し，収集した許可ログをポリシに変換する．その後，運用中のポリシと許可ログから変換されたポリシを比較し，運用中のポリシから不要なポリシを削除する．不要なポリシを発見する例を**図 3** に示す．この例では，運用中のポリシと許可ログから変換されたポリシを比較した結果，`write` を不要な権限であるとして，運用中のポリシから削除する．提案手法では，SELinux に許可ログを出力させるために，運用中のポリシに `auditallow` 文を追加している．

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（以降，アトリビュート展開）．アトリビュート展開の例を**図 4** に示す．許可ログから作成したポリシと運用中のポリシを比較する際に，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．これにより，粒度の細かいポリシ削減を実現している．

3.　提案手法

3.1　文献 [4] の手法の問題点

文献 [4] の手法には以下の問題点が存在する．

（問題点 1）　同一ポリシにより許可されたアクセスのログが出力され続けること

例えば，Apache において，`httpd_sys_content_t` というタイプが付与されたファイルに 10 回アクセスした場合，`httpd_sys_content_t` に対して `read` を行ったという許可ログが 10 回出力される．このように，同一ポリシにより許可されたアクセスのログが出力され続けるため，収集する監査ログのファイルサイズが巨大になる．これらの許可ログから変換されるポリシは 1 種類であるため，2 回目以降の許可ログの出力は不要である．

（問題点 2）　すべてのモジュールを対象にして提案手法を適用できないこと

（問題点 1）で述べたように，内容の重複するログが多く出力されるため，収集する監査ログのファイルサイズが巨大になる．文献 [4] において，4 つのモジュールに対して提案手法を適用し，2 日間ログを収集したところ，監査ログのファイルサイズは約 3.8GB になった．全部で約 400 個あるモジュールに対して提案手法を適用し，ログを収集した場合，収集する監査ログのファイルサイズが膨大になる可能性がある．このため，すべてのモジュールを対象にして提案手法を適用できない．

（問題点 3）　提案手法適用時のオーバヘッドが大きいこと

文献 [4] で行った ApacheBench を用いた評価では，apache モジュールに対して文献 [4] の手法を適用した際，ログ収集とポリシ削減期間において，約 164%のオーバヘッドが発生した．これは，出力されるログが膨大であることが原因であると考えられる．

3.2　対処

3.1 節で述べた 3 つの問題点を解決するため，我々は，監査ログの出力を抑えるように提案手法を拡張した．それぞれの問題点に対する対処を以下に示す．

（対処 1）　ログ変換部が変換したポリシのサイズが閾値を超えた際，一度ポリシの形式に変換されたアクセスルールに関する `auditallow` 文をポリシから削除すること

ログ変換部が変換したポリシのサイズが閾値を超えた際，一度ポリシの形式に変換されたアクセスルールに関する `auditallow` 文をポリシから削除することで，ログの出力を抑える．運用中のポリシに含まれる `auditallow` 文とログから変換されたポリシを比較し，一度ポリシの形式に変換されたアクセスルールに関する `auditallow` 文をポリシから削除する．削除する例を**図 5** に示す．図 5 に示す例では，ドメイン `foo_t`，タイプ `bar_t`，オブジェクトクラス `file`，アクセスベ

図 5　一度ポリシの形式に変換されたアクセスルールに関する auditallow 文をポリシから削除する例

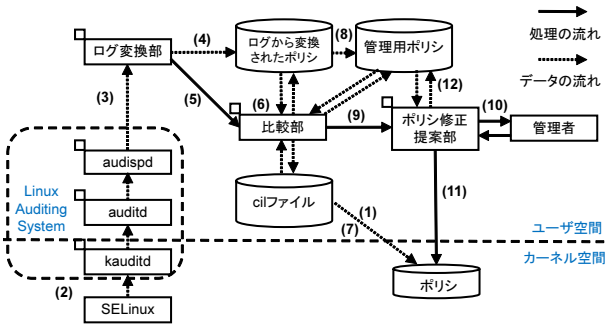

図 6　ポリシ削減機能

クタパーミッション read と write に関するアクセスルールが一度ポリシの形式に変換されているため，運用中のポリシから auditallow 文を削除している．このように，一度ポリシの形式に変換されたアクセスルールに関する auditallow 文をポリシから削除することで，削除後は出力されなくなるため，ログの出力を抑えることが可能である．（対処 1）により，（問題点 1）と（問題点 3）に対処した．

（対処 2）　audit dispatcher daemon の利用

ログのファイルサイズが膨大になるのは，監査ログの内容をすべて保存していることが原因である．文献 [4] の手法では，蓄積した監査ログをログ変換部がポリシの形式に変換していた．ログのファイルサイズが膨大になることには，audit dispatcher daemon（以降，audispd）を利用することで対処する．audispd とは，Audit デーモンからログを受信し，他のアプリケーションに送信するデーモンである．audispd からログを受信したログをポリシの形式に変換することで，監査ログの内容をすべて保存する必要がない．監査ログを蓄積する必要がないため，ログのファイルサイズは膨大にならない．（対処 2）により，（問題点 2）に対処した．

3.3　提案手法の処理流れ

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) 運用中のポリシ（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) ログ変換部が audispd からログを受信
(4) ログ変換部がログをポリシの形式に変換
(5) ログ変換部によって変換されたポリシのサイズが閾値を超えていた場合，比較部を起動
(6) 比較部がログ変換部によって変換されたポリシと運用中のポリシを比較し，一度ポリシの形式に変換されたアクセスルールに関する auditallow 文をポリシから削除
(7) 一度ポリシの形式に変換されたアクセスルールに関する auditallow 文を削除したポリシをシステムに反映
(8) ログ変換部が変換したポリシを管理用ポリシに保存
(9) （2）～（8）を一定期間繰り返したあと，管理用ポリシと 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
(10) 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
(11) 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
(12) 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

なお，(3)～(8) が文献 [4] の手法からの拡張部分である．

また，提案手法におけるポリシ復元機能の処理流れは，文献 [4] の手法の処理流れと同様である．

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，ポリシのサイズ，ラベルの数，モジュールの数，および allow 文の数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．

（評価 1）と（評価 2）において，評価対象の計算機では，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．各モジュールのログの収集期間は 2 日間とする．また，一度ポリシの形式に変換されたアクセスルールに関する auditallow 文は，ログ変換部によって変換されたポリシのサイズが 1MB を超えていた場合に削除するものとする．

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．httpd_config_t には 3 つの権限（search, open, getattr）を与えている．また，httpd_log_t, httpd_sys_contents_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．

4.3 ポリシ削減量

ポリシのサイズ，ラベルの数，モジュールの数，および allow 文の数を評価した．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．ポリシはカーネルにロードして利用されるため，ポリシのサイズの削減は，メモリ使用量の削減と対応している．また，ラベルの数，モジュールの数，および 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る．

全部で 413 個のモジュールのうち，削減対象としたモジュールは 388 個である．削減対象としていないモジュールは，base モジュールと監査システムに関するアクセスルールを含むモジュールなどである．base モジュールは，システムに必須のモジュールであり，容易に修正を行うものではないため，削減対象から除外した．また，監査システムに関するアクセスルールを含むモジュールに auditallow 文を追加した場合，ログが出力された際，ログの出力に伴ってさらにログが生成され，無限にログが生成される．これにより，システムが動作を停止する可能性があるため，削減対象から除外した．本評価の手順を以下に示す．

(1) 各モジュールに 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

表 1 に評価結果を示す． ここで，ポリシのサイズは，モジュールの削除や，利用しているモジュール内の不要なポリシ削減により，約 67%削減された．ラベルの数は，ラベルを定義しているモジュールの削除によって，削減された．allow 文の数は，不要なモジュールの削除や，利用しているモジュール内の不要なポリシ削減により削減された．

ラベルの数から，本環境では，ラベルの約 55%は実際には利用されていないことがわかる．また，モジュールの数から，モジュールの約 76%が実際には利用されていないことがわかる．これは，refpolicy が汎用的なポリシであり，利用されていないデーモンやアプリケーションに関するモジュールを多く含んでいるからであると考えられる．さらに allow 文の数から，本環境では，各モジュールで定義されている allow 文の約 81%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．

```
(allow httpd_t httpd_config_t (dir (search open getattr)))
(allow httpd_t httpd_log_t (dir (search getattr)))
(allow httpd_t httpd_sys_content_t (dir (search getattr)))
(allow httpd_t httpd_modules_t (dir (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 (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

表 1 ポリシの削減量

	デフォルト	削減後	削減量 (%)
ポリシのサイズ (B)	3,880,828	1,270,687	67.3
ラベルの数	4,833	2,180	54.9
モジュールの数	413	101	75.5
allow 文の数	103,693	19,629	81.1

表 2　4 つのモジュールにおける 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	85,694	91	85,603 (99.9%)
bind	211	4,649	59	4,590 (99.7%)
samba	1,088	34,181	97	34,084 (99.7%)
ssh	714	14,562	89	14,473 (99.4%)
合計	3,256	139,086	334	138,750 (99.8%)

表 3　すべ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 ($t1$)	許可ログ出力時（条件 1） ($t2$)	許可ログ出力時（条件 2） ($t3$)	オーバヘッド ($t2-t1$)	オーバヘッド ($t3-t1$)
11.579	57.111	11.519	44.532 (393.23%)	−0.06 (−0.52%)

また，HTTP サーバ，SFTP サーバ，SMB サーバ，および DNS サーバに対応する 4 つのモジュール（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）の allow 文の削減数を**表 2** に示す．

表 2 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 9 割以上の不要な権限を削減できたことがわかる．

以上のことから，提案手法により，各モジュールに含まれる不要なポリシを削減することで，ポリシのメモリ使用量を削減し，最小特権に近づけることができるといえる．

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ない期間と許可ログを出力している期間における Web サーバの性能を測定した．ここで，許可ログを出力している期間は，以下の 2 つの条件でそれぞれ測定した．

（条件 1）　auditallow 文を追加した直後

（条件 2）　一度ポリシの形式に変換されたアクセスルールに関する auditallow 文をポリシから削除した直後

性能を測定するために Web サーバに対するすべ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．本評価では，Web サーバは Apache 2.4.29，ベンチマークツールは ApacheBench 2.3 を用いた．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，すべてのリクエスト完了に要した処理時間の平均を算出した．クライアント側の環境は，CPU は Intel(R) Core(TM) i7-6700 3.40GHz，メモリは 8GB，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 44.532 秒 (393.23%) となった．文献 [4] での評価よりもオーバヘッドが増加している理由は，拡張した提案手法が audispd を利用しているためであると推察できる．ログ変換部までのログの送受信の時間により，オーバヘッドが増加したと推察できる．また，許可ログ出力時（条件 2）の測定結果からは，提案手法非適用時と同等の処理時間となっていることがわかる．これは，一度ポリシの形式に変換されたアクセスルールに関する auditallow 文をポリシから削除し，同一ポリシにより許可されたアクセスのログが出力されなくなったためであると推察できる．

ログ収集とポリシ削減期間中，文献 [4] の手法では，約 164%のオーバヘッドが常に発生する．これに対して，提案手法では，提案手法適用直後は，計算機の性能が落ちるものの，auditallow 文を削除する処理により，ログの出力が抑えられ，オーバヘッドが減少していくと推察できる．

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 [14] がある．文献 [14] は，すべ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 [15]，文献 [16]，および文献 [17] がある．文献 [15] 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 [16] は，拡張した `strace` コマンドを利用して，アプリケーションの振る舞いを調査する．調査した結果から，ポリシに含まれる不要なアクセスルールを削除することで，SELinux のセキュリティを強化する．しかし，この論文では有用性の評価に関しては述べられていない．文献 [16] は提案手法と似ているものの，ポリシ作成に必要な情報のすべてを取得したログから得られない点が異なる．

文献 [17] 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，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し，ポリシの開発に利用される．これに対して提案手法は，許可ログを利用して，既存のポリシから不要なポリシを発見し，削除することで，ポリシを最適化する．

6. おわりに

文献 [4] の手法には，同一ポリシにより許可されたアクセスのログが出力され続ける点，すべてのモジュールを対象にして提案手法を適用できない点，および提案手法適用時のオーバヘッドが大きい点という 3 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，提案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．拡張した提案手法は，ログ変換部によって変換されたポリシのサイズが閾値を超えた際，一度ポリシの形式に変換されたアクセスルールに関する `auditallow` 文をポリシから削除することで，ログの出力を抑える．これにより，3 つの問題点に対処した．また，拡張した提案手法により，最小特権に近づけられること，ログの出力を抑制できること，および利用されていないモジュールを発見し，削除できることの 3 つが期待される．

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，388 個のモジュールを対象として，ポリシの削減を行った．評価結果から，提案手法適用により，ラベルの数を約 55%，モジュールの数を約 76%，`allow` 文の数を約 81%削減し，最小特権に近づけることが可能であることを示した．また，不要なポリシを削減することで，ポリシのサイズを約 67%削減し，メモリ使用量の削減が可能であることを示した．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，提案手法適用した直後は，計算機の性能が落ちるものの，`auditallow` 文を削除する処理により，ログの出力が抑えられ，オーバヘッドが減少していくことを示した．

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 [2018 年第 2 四半期（4 月〜6 月）], 入手先 ⟨`https://www.ipa.go.jp/security/vuln/report/JVNiPedia2018q2.html`⟩（参照 2018-08-06）.

[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] 齋藤凌也，山内利宏：SELinux CIL を利用した不要なポリシ削減手法の提案，情報処理学会研究報告，Vol.2018-CSEC-82，No.30，pp.1–8（2018）.

[5] 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）.

[6] Richard Haines: The SELinux Notebook(4th Edition), available from ⟨`http://freecomputerbooks.com/books/The_SELinux_Notebook-4th_Edition.pdf`⟩ (accessed 2017-04-14).

[7] TresysTechnology: SELinux Reference Policy, GitHub, available from ⟨`https://github.com/TresysTechnology/refpolicy`⟩ (accessed 2017-01-12).

[8] 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).

[9] 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).

[10] 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).

[11] 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).

[12] 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）.

[13] 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).

[14] 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）.

[15] 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）.

[16] Marouf, S., Phuong, D.M. and Shehab, M.: A Learning-Based Approach for SELinux Policy Optimization with Type Mining, Proc. 6th Annual Workshop on Cyber Security and Information Intelligence Research (CSIIRW'10), Article No.70, p.4 (2010).

[17] 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).