SELinux のポリシ作成時間を短縮する一考察

桝本　圭†　村田　裕之‡
E-mail：†masumotok@nttdata.co.jp, ‡murata@ndc.ne.jp

概要

セキュア OS は、様々な OS に標準搭載される程に身近な技術となり、その中でも SELinux は最もメジャーなセキュア OS である。しかし、SELinux の設定（アクセスポリシ）の作成は、非常に時間のかかる作業である上に、設定ミスがあると、OS 上で動くプログラムの動作妨害となるため、ビジネス用途では敬遠されがちである。そこで、これら二つの課題を軽減する検討を行った。検討の結果得られた手法とテスターを用いた効果測定結果について紹介する。

１．はじめに

セキュア OS は、通常の OS と比べて細かいアクセス制御を行う OS であり、主に不正アクセス対策や、管理者の誤操作防止を行うために適用される。

その特徴は二つあり、一つ目に、通常の OS だと管理者はアクセス制御を無視してシステムを操作できるが、セキュア OS だと管理者でもアクセス制御には従わなくてはならない。そのため、不正アクセスにより管理者権限を奪取されると、通常 OS では Web コンテンツの改竄や踏台の設置等の被害を受けるが、セキュア OS ではこれらの被害を最小限に抑止できる。また、バックアップデータを消去してしまうような管理者の誤操作も禁止できる。

二つ目の特徴は、通常の OS より細かいアクセス制御である。例えば通常の OS ではアクセス制御はユーザ・グループごとの設定が基本だが、セキュア OS ではプログラム単位でアクセス制御を行うため、必要最低限のアクセス権限を与えやすい。

このような利点を元に、セキュア OS は、各種 OS に搭載されるようになり、非常に身近な技術となった。そのうち最もメジャーなものとして SELinux[1] がある。

SELinux は、多くの Linux に標準搭載され、OS インストール時に SELinux の機能が有効となるため、大多数の Linux ユーザが使う機会がある。

しかし一般に、導入には相応の工数が必要とされ、ビジネス用途では敬遠されがちである。その原因として想定されるものを以下に示す。

(1) 動作保障のための試験工数の肥大化 ― セキュア OS 導入時には、システム管理者がアクセスポリシ（以降、ポリシ）と呼ばれる設定を記述する。ポリシとは、多くの場合、プログラムがアクセスしてよい OS リソースを定義するものであって、定義が不十分だとプログラムが動かないため正確に記述する必要がある。しかし、プログラムがアクセスする OS リソースを正確に把握することは難しい。一般には、様々な試験によりプログラムを動かし、アクセスされたリソースをログに出力することが行われるが、どのように動かせば全部のアクセスが列挙できるか不明であるため、多くの時間を要する。このような試験は、拡張モジュールの導入、大規模なパッチ適用、システム構成の変更の際にも発生し、また高い信頼性が求められる場合、システム異常系の動作も試験対象となるため、動作保証のための工数は少なくないといえる。

(2) 独自概念の習得に要する時間 ― 一般にセキュア OS のアクセス制御は、通常 OS のアクセス制御とは異なる概念で実現され、理解に時間がかかる。さらに SELinux の場合は、ポリシ作成用の独自言語を採用しており、習得知識はさらに増えている。現状 SELinux では、apache 等主要なプログラムのサンプルが配布されており、ユーザが自作のポリシを作成する機会は減少しているが、独自開発されたプログラムやサンプルポリシがない場合は、ユーザは自力で対応しなければならない。

†株式会社 NTT データ 技術開発本部
Research and Development headquarters, NTTDATA Co.
‡日本データコム株式会社 IT ソリューション事業本部
IT solution headquarters, NIPPONDATACOM

(3) 設定内容の理解が困難 ― SELinux の場合、ポリシは独自言語で記述されるため、内容が理解しづらい。そのため、作成したポリシを第三者がレビューしづらく、また、ベンダ提供のサンプルポリシを使う場合も同様で、ユーザのシステム構成も想定されているか容易に判断できない。ポリシの不備がシステムの動作妨害を伴う危険性の元では、ある程度、ユーザがポリシに関して納得感を持てないと、不安が生じる要因となる。

そこで本稿では、これらの課題を考慮して、多くの知識を必要とせず、時間をかけずセキュア OS を導入できることを目的に行った検討内容について述べる。これにより、これまでハイスキルな技術者が時間をかけて導入していたセキュア OS の敷居を下げ、より多くの利用者がセキュア OS の利点を享受するための環境整備につなげる。

2. SELinux のアクセス制御（用語の定義）

SELinux は、TE、RBAC、MCS、MLS というアクセス制御を採用している。このうち、現状 MCS、MLS については、実質 SELinux に精通するユーザ向けという位置づけとなるため、本稿の対象外とする。

TE、RBAC のうち、TE ではプロセスのアクセス制御を行う。例えば、/etc/init.d/httpd が /usr/sbin/httpd を実行した場合には、権限 A が与えられ、/etc/logrotate.d/httpd が /usr/sbin/httpd を実行した場合は別の権限 B が与えられる。一般には「プロセス X が実行したプログラム Y には、Z という権限が割当てる」形式である。

このようなアクセス制御は、アクセスする主体であるプロセスや、アクセスされる客体であるファイル等について、類似のものをグルーピングした後に行う。図 1 は一例であり、/etc/init.d 以下にはデーモンプログラムを起動するスクリプトがあるが、それらは通常一つのグループになり、initrc_exec_t という名称がつけられる。/sbin/init が initrc_exec_t グループに属するプログラム（/etc/init.d/xxx）を実行すると、実行したスクリプト毎に起動するプロセスの種類は異なるが、それらは一律グルーピングされ、initrc_t という名称が付与される。また、/sbin/init についても init_t というグループに属している。これらのグループ情報はポリシに記述される。

なお、initrc_t のようにプロセス等の、主にアクセス主体となるグループを「ドメイン」、ファイル等のようにアクセス客体となるグループを「タイプ」と呼ぶ。また、init_t ドメインが initrc_exec_t を実行することにより、他ドメインである initrc_t に移ることを「ドメイン遷移」と呼ぶ。タイプについて、initrc_exec_t のように、特にドメイン遷移の契機となるタイプを「エントリポイント」と呼ぶ。

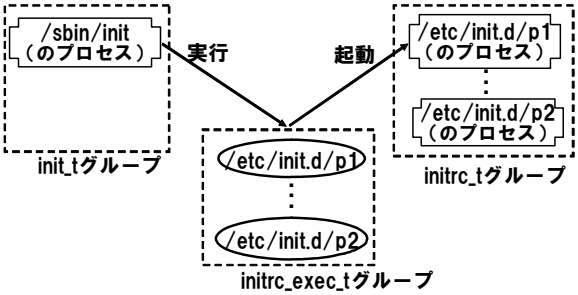

図 1 SELinux が採用するアクセス制御

```
1: allow initrc_t inirc_exec_t:file execute;
2: type_transition httpd_t var_log_t:file httpd_var_log_t;
```

図 2 代表的なポリシの構文

```
1: user webmaster roles webmaster_r;
2: roles webmaster_r types httpd_t;
   roles webmaster_r types webmas_t;
```

図 3 RBAC 関連のポリシ構文

図 2 はこれらのアクセス許可の一例である。図 2 の 1 は、ドメイン initrc ドメインが initrc_exec_t の file に対して execute できる意味となる。一般には「ドメイン A はタイプ B のリソース C に D のアクセスを可能とする」という意味になる。このうち、C のことをオブジェクトクラスと呼び、ファイル、ディレクトリ、FIFO 等 50 以上存在する。また、D のことをパーミッションと呼び、基本的にはシステムコール単位で約 200 種類存在する。

また図 2 の 2 は、「httpd_t が var_log_t タイプのディレクトリ以下にファイルを作成する場合、そのファイルは httpd_log_t グループとなる」意味となる。基本的に、あるドメインがあるディレクトリ以下にファイル等を作成した場合、ディレクトリのタイプが継承される。しかし/var/log 等、様々なプログラムから共有されるディレクトリ以下に配備されるリソースについては、他とは別タイプを割当てることが慣例である。こうすることで、他のプログラムのログファイルとは別権限を

割当てることが可能となり、例えば httpd_t は /var/log/httpd(httpd_log_t)には読書きできるが、/var/log/mysql.log (mysql_log_t：他プログラムのログ) にはアクセスできない状況が実現できる。

次に RBAC とは、ユーザのアクセス制御を行うものであり、管理者権限を分割する目的で使う。例えば、Web サーバの管理ユーザ webmaster を定義し、ログイン後に、Web サーバの起動停止ができるといったように、管理者 (root) 以外のユーザに管理者権限を譲渡することに使う。

SELinux では、あるプログラムが実行されてできるプロセス群のアクセス権限をドメインとして定義するため、これを用いて、様々なプログラムを実行するであろうユーザのアクセス権限は、基本的には任意の数のドメインを一まとめにして、ユーザに使用させればよい。正確には、図3の1のように、ユーザにロールと呼ばれる役割を割当て、ロールごとに使用できるドメインを決める。例えばユーザ webmaster が、ログイン直後のドメインと、Web サーバプロセスのドメインを使える必要があるとすると、まず webmaster_r というロールを割当て、次に図 3 の 2 のように webmaster_r が二つのドメインを使えるように設定する。

3．課題の分析と対応指針

1 章で述べた課題のうち、プログラムの全てのアクセス範囲を抽出するための試験工数を要する課題については、SELinux が「明示的に記述したアクセスのみ許可される」ことを前提とする以上改善は難しい。その理由を説明するため、現在 SELinux が採用する Reference Policy[2] を例に挙げる。これは、あるアクセスを許可する際に、関連するアクセスも同時に許可するものである。

図 4 は一例であり、Web サーバ (httpd_t) が bin_t (/bin 以下のコマンド) を実行するための記述である。ユーザが、図 4 の (a) ようにマクロでポリシを記述すると、内容は (b) であるため、 (a) を展開すると (c) となる。つまり、Web サーバを動かしてみて、 (c) の 3 行目に記載されたアクセスが確認できたなら、1 行目、2 行目のポリシも一緒に記述すべきことを示している。

このことからわかるように、Reference Policy は、 (c) の 1、3 行目は記述したが、2 行目のアクセスは確認できなかったために、アクセス許可を記述しなかったという記述漏れは補完できる。だが、 (c) にあるアクセスのうち、いずれも確

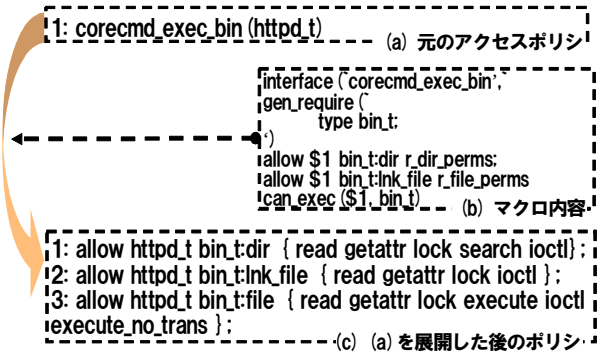

図 4 Reference Policy の例

認できなかった場合を考えると、厳密な試験が必要である状況に変わりがなく、既存技術では課題の解決は難しいことがわかる。

そこで、従来のポリシ作成法とは逆の方法により改善を行う。すなわち、初めは全リソースに関してアクセス許可であり、アクセスを制限するところを指定させる。このアプローチにより、「明らかにプログラムがアクセスしないところは制限する。不明なところは制限しない」ことが可能になり、ポリシの作成ミスにより、プログラムの動作妨害をする可能性は減少する。

また、このアプローチだけで十分ではない。SELinux では、2 章で述べたオブジェクトクラスが 50 以上、パーミッションが約 200 存在し、ユーザがこれらを使い分ける必要があり、これらを無理して使い分けたところで、プログラムの挙動が少し変更されただけで、ポリシ修正や再試験が必要となる。そこで表 1 に示すように、まとめて設定させることにする。

表 1 のうち、項番 6 については、利用するケースがさほど多くないこと、また、プログラムがある一つのアクセスを許可されるためには、複数のチェックが行われることが多いため、他の項番のオブジェクトクラスを制御しておけばよいと判断した等の理由から、制御しない (全てのアクセスを許可する) ものとした。項番 3 のネットワークポートについては、設定の簡易性を考え、使用するポート番号を指定する。他の項番のように、使用しないものを指定する形式ではない。

またパーミッションについて、特に項番 1 については、まず UNIX 系 OS と同じように rwx (読み、書き、実行) で検討した。しかし、w は書込み権限だけでなく、リソースの作成／削除権限も付与されるため、作成／消去の許可 (c：create) は

表 1 オブジェクトクラスの種類と制限

項番	項目	オブジェクトクラス代表例	制限の種類
1	ファイルシステム上に存在するリソース	file, dir, fifo_file, chr_file, blk_file, unix_stream_socket	r, w, x, c4種類の組合わせ
2	ケーパビリティ	capability	指定したケーパビリティの設定ができないようにする
3	ネットワークポート	tcp_socket, udp_socket	あるポート番号にbindできる／できない
4	inetドメインソケットを用いたネットワーク通信	tcp_socket, udp_socket	ソケットを用いた通信ができる／できない
5	シグナル	signal	SIGCHLD, SIGKILL, SIGSTOP, SIGNULL, それ以外の4種類のシグナルが送れる／送れない
6	その他(1)	security, raw_socket, passwd, isem, msg, msgq, shm, ipc	制御しない

分離して、rwxc の 4 通りの組合わせとした。これは、例えば/dev 以下のリソースに書込むケースはあるが、作成／消去するといったケースは少なくないと判断したためである。

またその他の課題である、独自概念の理解に時間を要することについては、前提知識がなくとも設定を可能とする GUI を提供することで軽減が可能であり、また設定内容の理解が困難である課題には、独自言語を使用せず、ユーザが比較的慣れた形式で可視化することで軽減できる。これらは 4 章で画面イメージと共に説明する。

4．プロトタイプ

4．1　プロトタイプの概略

3 章の指針を元に、表 2 の OS に標準搭載される SELinux を対象に、python2.5、wxPytyon2.6.3.3 等の言語を用いてプロトタイプを実装した。プロトタイプは WindowsXP で動作し、設定対象 OS に ssh 経由でサーバに接続する形式で動作する。

また現状 SELinux の主要な使用方法は、インターネットサーバの不正アクセス対策である。そこでプロトタイプでは、サーバプログラムのアクセス制御を想定しており、ユーザは対象外とした。

4．2　本手法によるアクセス制御の基本

指定した箇所のみアクセス制限を実施するという基本指針を達成する上では、既存技術を生かす観点から、SELinux 自体の改良をしない方針とした。実際には、はじめに「あらゆるアクセスを許可する」記述を行っておき、ユーザによって指定された箇所のみアクセス制限を行う。

図 5 はその一例であり、 (a) ではあらゆるアクセスを許可し、 (b) では/bin 以下のリソースについて、読込みと実行のアクセスに制限している。

表 2　評価用プロトタイプで用いた OS（サーバ）

	プロトタイプ1	プロトタイプ2
OS	RedHat Enterprise	Debian Sarge
カーネル	2.6.9-5.EL	2.6.12
SELinuxポリシ	selinux-policy-targeted-1.17.30-2.52.1	selinux-policy-1.26-7
libselinux	libselinux-1.19.1-17	libselinux-1.30-1

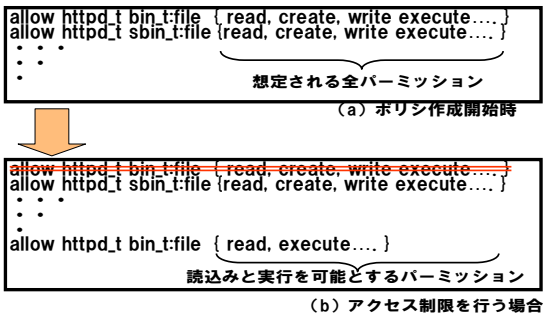

図 5 本手法によるアクセス制限の基本

4．3　ポリシ作成の流れ

ポリシ作成の流れを図 6 に示す。初めに、これからポリシを作成するプログラムの指定と、そのプログラムを実行するユーザ（ロール）を指定する（エントリポイントの指定）。次にアプリケーションの種類によらず、アクセス制限すべき項目を設定し（共通設定）、その後でポリシを作成するプログラム固有のリソースについてアクセス制限を行う（個別設定）。最後に、4.6 節で後述するプログラム間の相互影響を解消し、作成したポリシを SELinux に読込ませる。

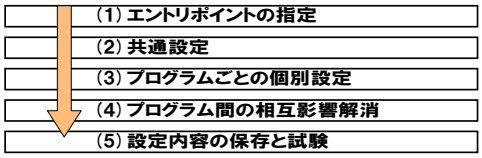

図 6 全体設定フロー

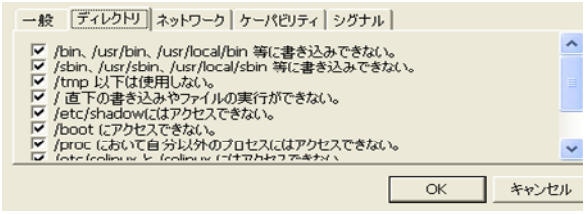

図 7 設定テンプレート

4．4　テンプレートを用いた共通設定

エントリポイントの指定後は、図 7 に示すテンプレートによりポリシ作成を開始する。

このテンプレートは、Unix 系 OS の FHS(File System Hierarchy)[3]に従うプログラムであれば、共通にアクセス制限すべき箇所を提示する。例えば、図 7 の「/bin, /usr/bin 以下のコマンドを・・」は、コマンド改竄やバックドアを設置する行為を禁止する項目だが、通常のプログラムはこのような行為を行わないため、テンプレート通りに設定しても、動作妨害が発生しにくい。こういった、動作妨害は発生しにくいが、主要な不正アクセスの被害を防止できる観点に基づきテンプレートは提示される。したがってユーザは、深い考察なしで大方のポリシを作成することが可能である。なお、設定内容は図に述べる GUI に反映されるため、その効果が一目でわかるようになっている。

4．5　個別設定

テンプレートによる設定後は、まだ未設定の箇所について考える。それらは主に、Web サーバにおける Web コンテンツのように、プログラムの種類毎に異なるリソースである。

設定時には、1 章で述べるように、多くの前提知識を必要とせず、設定内容の全体俯瞰ができることが重要である。そこで、システム管理者が慣れているエクスプローラ形式の GUI に基づき設定する。図 8 はその例であり、画面に表示される各ファイルやディレクトリを指定すると、図 9 の画面が起動し、設定ができる。また設定内容は、図 8 のように、設定内容が一目で判断できる。また SELinux

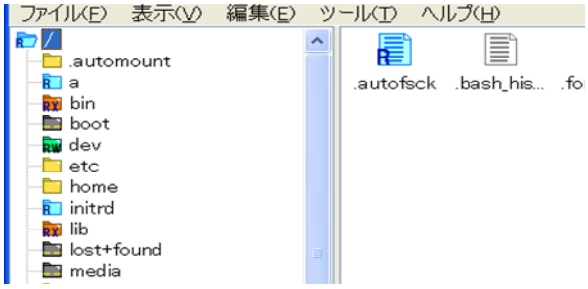

図 8 設定内容の可視化画面

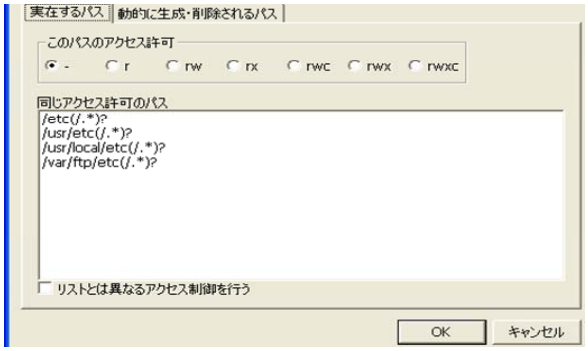

図 9 アクセス制御設定画面

では、あるプロセス（ドメイン）のアクセス権限を、個別のファイル単位ではなくタイプ単位に与えるため、例えば/etc/init.d/httpd に対してアクセス許可を与えた場合、同じタイプに属するリソースにもアクセス権限が付与されるが、ユーザは GUI で逐次確認できる。また、図 9 の画面でも、/etc/init.d/httpd と同じグループ（タイプ）に属するリソースを参考情報として表示している。

なお、図 9 下部の「リストとは異なるアクセス制御を行う」とは、新しいグループ（タイプ）作成を行うときに用いるオプションである。

4．6　ポリシ間で発生する相互影響の解消

現状 SELinux は、OS が通常稼動するためのポリシが既にある状態でユーザに提供される。ユーザは、基本的には、アクセス制限したい個別のプログラムのポリシを作成する。本稿で述べる手法も同様である。このとき、新たに二つ検討すべき課題が生じたため、その内容と対処について述べる。

(a) 他プログラムとの依存関係

プログラム A、B と、本稿で述べる手法でこの順に作成されたアクセスポリシ P_A、P_Bがあるとする。このとき、例えば P_Aにおいて/etc/init.d/httpd につ

いてアクセス制御を行う。このとき、SELinux では既に/etc/init.d 以下は initrc_exec_t というグループ（タイプ）に分類されるため、initrc_exec_t へのアクセス制御となる。

次に P_B において、/etc/init.d/mysql のみ新しいタイプを割当て、アクセス制御を行ったとする。つまり、/etc/init.d 以下は、initrc_exec_t タイプであるが、/etc/init.d/mysql だけは別タイプとなる。

この新しいタイプは、P_B で設定したものであるが、SELinux の仕様により、P_B が設定したタイプ分類に P_A も従わなければならない。つまり、P_A 作成時点では/etc/init.d 以下全てにアクセス制限を行ったはずが、P_B 作成後は、/etc/init.d/mysql にはアクセス制限をしていないことになる。このような状況が発生した場合、ユーザに通知する必要がある。

(b) 従来方式によるポリシと本方式によるポリシの共存

Web、DB サーバがあり、Web サーバから DB サーバ内の情報を検索する要求を出す。Web サーバのポリシ P_{WEB} は、指定した箇所のみアクセス許可する従来手法で既にあるものを使い、DB サーバは、本稿で述べる手法でこれからポリシを作るとする（P_{DB} とする）。

このとき、P_{DB} を作成後、P_{WEB} にも変更が必要となる。Web サーバは DB サーバに情報検索要求を出す（DB サーバが提供するポートに接続する）ことを許可する記述が必要だからである。

これらの問題は「二つのアクセスポリシ P_A、P_B をこの順に作成した。後から作成した P_B が先に作成した P_A に影響を与えてしまう」とまとめられる。ここでは (a) 、 (b) 共に、P_B 作成終了時に、図 10 の画面を提示することで対処する。

図 10 では、「変更するアクセス制御一覧」以下に、P_B が P_A に影響を与えるリソースを全て抽出して表示している。 (a) の例で言えば、P_B において新タイプを定義した時点で、新タイプ定義前のタイプ名を覚えておき、P_A でそのタイプが使用されていないか検査する。

また (b) の場合、P_{DB} 作成後に図 10 を開くと、P_{DB} で定義した新タイプが全て表示する。そこで Web サーバについて、P_{DB} で定義された DB サーバのポートに関するアクセス制御を設定する。

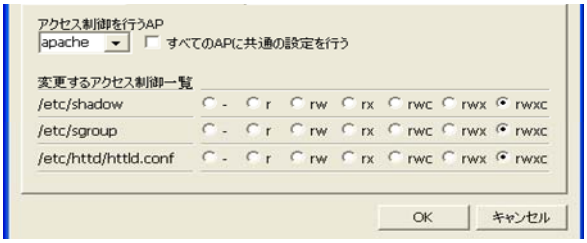

図 10 依存関係の解消

５．効果測定

５．１ 測定方法

Debian で構築されたある実システムにおいて、指定した箇所のみアクセス許可する従来手法と本稿で述べる改善手法でポリシを作成し、作成時間を計測することで、改善手法の効果を測定した。測定は、以下の条件を持つテスターが行った。

(a) 測定対象システムは、以前に SELinux を導入しており、テスターは導入チームで作業した経験がある
(b) 測定前に、試験回数により時間の差が生じないよう、SELinux 導入時の試験を練習した

なお作成時には、以前に測定対象に SELinux を導入した際に見積もった効果の代表例である表 3 の要件は最低満たすものとし、両者同程度の効果を持つアクセス制御を行わせるよう留意した。

また計測する時間とは、ポリシ作成からシステムの動作保障ができる程度の試験が終了するまでとした。試験項目は、比較的軽めにマニュアルに記載された全ての動作（試験項目数は 238）とした。

５．２ 測定対象のシステム

図 11 の (a) は、測定対象の概要である。主に開発を担当する部署で使われる Web システムであり、掲示板機能等大きく 4 つの機能が提供される。

このシステムでは、 (a) の一番右にある機密情報（ファイル）を管理する機能があり、システムが不正アクセスされた場合もこの情報を漏洩させないことが SELinux 導入の第一の目的である。

通常の OS であれば、どの機能が不正アクセスされた場合でも、管理者権限（root）を奪われれば、攻撃者に機密情報にアクセスされてしまう。

SELinux を導入すれば、図 11 の (b) に示すように権限を定義し、機密管理機能以外は機密情報にアクセスできないように設定できる。つまり、

表 3 ポリシ作成を行う上での最低要件

項番	要件
1	機密情報には、機密管理機能以外からアクセスできない
2	各機能のCGI(*.php等)やカーネル、/bin、/usr/bin、/sbin、/usr/sbinにあるバイナリの改竄ができない
3	スケジュール管理機能、機密管理機能からはWebメールが保持するメールデータにアクセスできない
4	ユーザのホームディレクトリ/home以下にはアクセスできない(機密が/home以下に格納されるため、機密管理機能は例外)
5	/var/log, /tmp等の共有ディレクトリについて、本システム以外のプログラムが作成したリソースにはアクセスできない
6	他のプログラムに対してシグナルを送れない
7	カーネルモジュールをロード/アンロードできない
8	SELinuxのアクセスポリシの変更ができない
9	/dev以下のデバイスファイルの改竄ができない
10	(主にデーモンのブートスクリプトである)/etc/init.d/以下にはアクセスできない

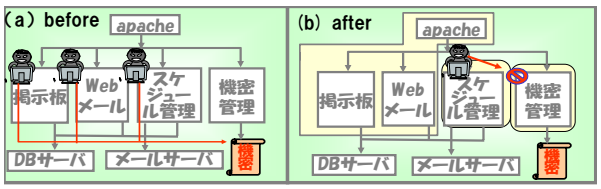

図 11 効果測定対象システム

機密管理機能が攻撃されて、管理者権限を奪われない限りは、機密情報にはアクセスできず、漏洩のリスクを減少することができる。

5．3 測定結果と考察

測定の結果、従来手法が 870 分、本稿で述べる手法では 385 分かかった。従来手法と比較し、56%の時間短縮ができたと言える。

図 12 にテスターの作業フローを示す。対象システムは、deb パッケージで構成されるため、まずテスターは dpkg コマンドでシステム固有のリソースを確認した。次に、表 3 の要件を満たすポリシ作成指針を作成した。ここまでは、両者共通作業のため、テスターはまとめて作業を行った。

次に従来手法の場合は、ポリシ作成指針を SELinux のポリシに直し、不足する記述がないか、試験と同様の作業による「情報収集」作業を行い、この結果を受けてポリシ記述を追記し、最後に試験を行った。なお、permissive や enforcing は SELinux の用語で、アクセス違反発生時に、アクセス禁止はしないがログを出力する確認モード、アクセス禁止を行う運用時動作モードの意味である。また改善手法の場合は、4 章で述べたプロトタイプを用いて設定した後で試験を行った。

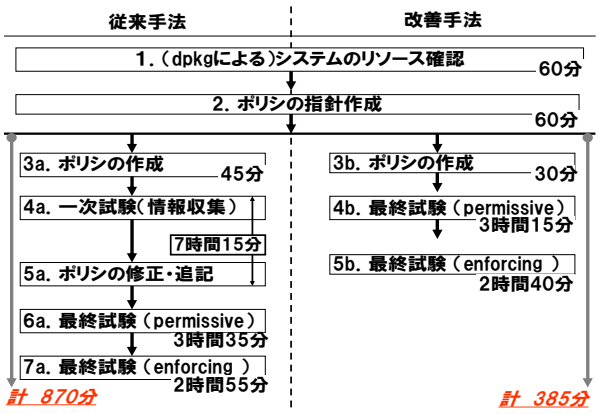

図 12 測定におけるテスターの作業フロー

図 12 より、二種類のポリシ作成で時間差が発生した箇所は、一つ目に 3a と 3b であり、これはテンプレート等の効果であると推定できる。二つ目に改善手法では、一番時間がかかっている 4a に相当する作業がない。これは従来手法のように、アクセス許可漏れが発生しないように、プログラムのアクセス範囲を抽出する必要がないためである。

このように、本稿で述べる改善手法は、ポリシの作成時間を短縮するものであると言える。

6． 考察

6．1 手法のアプローチに関する利点と欠点

改善手法では、ポリシ作成時間短縮の利点を提供する。前章で示した効果に加え、改善手法では、以下の点から必要な前提知識の量と作業量がより少なく済むことが期待できる。

従来手法は、一旦プログラムにアクセス制限をかける場合、制限内容に関わらず、プログラムの全アクセス範囲を抽出する作業と、プログラムの全挙動を理解することが必要である。

しかし改善手法では、図 11 のシステムを例に説明すれば、SELinux を導入する上では、表 3 の項番 1 が最低条件であった。これを実現するには、図 11 の機密管理機能以外が、機密情報にアクセスできない制限を設定するだけでよい。つまり、機密情報のファイル名と、アクセスさせたくないプログラム名を知っておけばよい。また、とりあえず表 3 の項番 1 のみ満たし、後日他の箇所もアクセス制限を行うという柔軟な設定も可能となり、セキュア OS の利点を享受しやすくなる。

逆に改善手法の欠点とは、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点である。4.4 節で述べるテンプレート等や GUI により、漏れが極力発生しないように配慮するが、それでも完全に漏れが起きないことは保証できない。

しかしながら、ポリシの安全性と、ポリシ作成に要する時間はトレードオフの関係にあり、一概に欠点とも言い切れない。アクセス制限の厳しい、すなわち安全性の高いポリシを作成するためには、プログラムの全アクセス範囲を厳格に抽出する手間もあれば、プログラムに追加変更に対して、すぐにポリシの修正が必要となる手間もある。アクセス範囲の抽出漏れにより、プログラムが突然動かなくなる可能性も否定しきれない。したがって、どちらの手法を取るかはユーザの選択であるし、改善手法は、従来手法との共存が可能である。

６．２ 効果測定に関する考察

5 章で述べた測定の結果、従来手法と改善手法では、485 分の差が生じている。この差を多いか少ないかと感じるかは、個人差があるであろう。しかし今回の測定は、試験項目を比較的軽く見積もり、かつ SELinux に精通した技術者が測定する前提に基づく。したがって、状況によりこの時間差は広がることが想定される。例えば、SELinux に不慣れな測定者なら、3a や 5a において差が拡大する。また、試験もより実際にあわせれば、試験作業の慣れ、試験項目の増加から、まず試験を一度通す時間が拡大し、4a でより時間がかかる。これは、自システムにおいて、各モジュールを結合した状況での総合試験を一度実施するのに要する時間を考えると、容易に想定できることであろう。

また、改善手法を使えば、要件に応じてより 4b、5b の作業を短縮させることも可能である。例えば、Web サーバが Web コンテンツを書き換える処理は通常行なわない。このように明白な制限しかしない場合は、試験の削減を検討してもよい。

それゆえに、今回の測定は、SELinux に精通する技術者であり、かつシステム操作に熟練した技術者でも発生する時間差の参考例と考えられる。

７． 関連研究

SELinux のポリシに関しては、既にいくつか取組みがある。SEEdit[4]はポリシ作成を簡易に行うものであり、TOMOYO Linux[5] や AppArmor[6] 等のセキュア OS は、SELinux より簡単に使える点を強調する。

これらの取組みは、明示的に設定したアクセスのみ許可という方針を採る点で類似する。本稿で述べる手法とは、指定したアクセスのみ制限する点で、基本的なアプローチが異なる。これらの手法とは、時と場合に応じて共存できる技術である。

また LIDS[7]は、明示的に設定したアクセスのみ許可の方針を採る点で類似するが、根幹となるアクセス制御方式に、ドメイン等の概念を持たない。そのため、SELinux をベースに実装する場合では、4.6 節で述べる内容が必要になる等の違いがある。

おわりに

SELinux のポリシの作成時間を短縮するための検討内容について述べた。ここでは、指定した箇所のみアクセスを許可する方式であった従来と比較し、指定した箇所のアクセスを制限する手法について述べ、また、使用できるオブジェクトクラスやパーミッションを制限することで、さらにポリシ作成時間の短縮を目的としたものである。また、厳しくアクセス制限しすぎないことで、プログラムの動作妨害が発生する課題を軽減する。

さらに実装したプロトタイプは、設定内容の可視化機能とテンプレートにより時間を短縮する工夫も行った。その結果、テスターによる効果測定では、従来手法と比べ、56%の時間短縮を確認した。

なお、今後の予定としては、効果測定の事例を増やしていくことがある。また、プロトタイプの公開については今後検討を行っていく予定である。

参考文献

[1] SELinux, http://www.nsa.gov/selinux/

[2] Reference Policy, http://oss.tresys.com/projects/refpolicy

[3] Filesystem Hierarchy Standard, http://www.pathname.com/fhs/

[4] 中村 雄一, 鮫島 吉喜, “Security-Enhanced Linux のアクセス制御ポリシ設定の簡易化,” 暗号と情報セキュリティシンポジウム, Jan. 2003

[5] 原田 季栄, 保理江 高志, 田中 一男, “TOMOYO Linux – タスク構造体の拡張によるセキュリティ強化Linux,” Linux Conference, Jul. 2004

[6] App Armor, http://ja.opensuse.org/AppArmor

[7] LIDS, http://www.lids.org

[8] セキュアなインターネットサーバ構築に関する調査, http://www.ipa.go.jp/security/fy14/contents/trusted-os/guide.html