

第 編 環境構築検証編

目次

1. 概要	1
1.1. 調査概要	1
1.2. 調査期間	1
1.3. 調査対象	1
1.4. 調査方法	1
2. 調査内容	2
2.1. SELinux	2
2.1.1. はじめに	2
2.1.2. SELinux の概要	2
2.1.3. 前提条件	4
2.1.4. SELinux カーネル構築方法	5
2.1.5. SELinux のインストール	11
2.1.6. SELinux の設定方法	28
2.1.7. SELinux の設定例	68
3. 付録	110
3.1. デフォルトマクロ一覧	110
3.2. デフォルト属性一覧	114

1. 概要

1.1. 調査概要

本報告書は、Security-Enhanced Linux (SELinux) の構築方法をまとめたものである。

1.2. 調査期間

本報告書の調査期間は、平成13年11月から平成13年12月である。

1.3. 調査対象

表 1.3-1 にあげるシステムを対象として調査を実施した。

表 1.3-1 環境構築調査システム

#	調査対象システム 【URL】
1	Security-Enhanced Linux (SELinux) 【 http://www.nsa.gov/selinux/ 】

1.4. 調査方法

対象システムを実際に構築し、以下の観点から調査を行った。

- カーネル構築方法
- インストール方法
- ポリシー設定方法

2. 調査内容

2.1. SELinux

2.1.1. はじめに

本構築調査において利用した SELinux のバージョンは以下のとおりである。

表 2.1-1 SELinux のバージョン情報

リリース時期	LSM バージョン	コメント
2001/12/10	lsm-full-2001_12_10-2.4.16	SELinux の 4th パブリックリリースに対していくつかのアップデートが施されてリリースされたバージョン。

また、構築調査において利用した Linux ディストリビューションは RedHat Linux 7.2(FTP 版)である。

2.1.2. SELinux の概要

SELinux とは、Linux カーネルに対してセキュリティ拡張したものである。米国家安全保障局 (NSA¹) で開発されており、GPL²で配布されているフリーソフトである。

SELinux は、Flask³と呼ばれるアーキテクチャを実装して TE⁴、RBAC⁵および MLS によるアクセス制御を行なう。本調査時点で MLS は実験的サポートという位置付けである。SELinux においてサブジェクトとオブジェクトにはセキュリティコンテキストと呼ばれるラベルが付与される。セキュリティコンテキストはユー

¹ National Security Agency

² GNU General Public License

³ Flux Advanced Security Kernel

⁴ Type Enforcement

⁵ Role-Based Access Control

ザ名、role、domain(type)およびMLSのレベル(MLSを有効にした場合のみ)を示すラベルによって構成される。サブジェクトからオブジェクトへの、あるいは、サブジェクト間のアクセスはこのセキュリティラベルに基づいて許可されなければならない。また、これらのアクセス制御で許可されているアクセスならば、通常のLinuxで実装されているファイルパーミッションによるアクセス制御も行なう。

SELinuxは、LSM⁶による実装とLSMによらない実装の2種類のソースが提供されている。現在はLSM版の開発のみが行われている。そこで本報告書では、LSM版SELinuxでのインストール方法および設定方法について記述する。

LSM版SELinuxはLSM、基本コマンド群およびSELinuxのために拡張されたデーモンなどのユーティリティからなる。基本コマンドには既存のLinuxコマンドを拡張したものと独自のコマンドが含まれている。

本調査時点でSELinuxは以下の配布形態をとっている。

1. LSMを含むLinuxカーネルファイルとSELinuxで必要となるユーティリティをまとめたファイル。
2. LSMを含むLinuxカーネルファイルと、SELinuxで必要となるユーティリティを別々のファイルにしたもの。
3. LSMパッチのみとSELinuxで必要となるユーティリティを別々のファイルにしたもの。
4. LSMパッチ、SELinuxで拡張されたユーティリティのパッチを別々のファイルにしたもの。
5. 上記の項番4をまとめて1つのファイルにしたもの。

⁶ Linuxシステムにおいて、セキュリティ拡張パッケージをサポートするためのフレームワークとなるロードブルカーネルモジュール。<http://lsm.immunix.org/>

2.1.3. 前提条件

2.1.3.1. 前提条件

SELinux をインストールする際の前提条件として、インストール先のマシンに Linux がインストールされている必要がある。ファイルシステムに関しては、Linux で使用できるファイルシステムならば SELinux でも使用できる。また、SELinux は一部のディストリビューションを除いて（現時点では Debian）バイナリ形式では提供されていない。そのため、カーネル構築および SELinux ユーティリティのコンパイルに必要な gcc や make などのソフトウェア開発環境ツール、および、SELinux 用に拡張された OpenSSH をコンパイルする際に libwrap.a ライブラリが必要になる。どちらともバージョンの指定はない。また SELinux の設定ファイルは RedHat Linux をベースとして作成されているので、RedHat Linux 以外で利用する場合には、使用しているディストリビューションに合わせてインストール時に設定ファイルなどを変更する必要がある。

2.1.3.2. RedHat Linux7.2 での前提条件

SELinux を RedHat Linux7.2 にインストールする場合の前提条件を説明する。RedHat Linux7.2 のインストール時に SELinux コンパイルで必要となるソフトウェアをインストールするため、「インストール種類」でそれぞれ以下のように選択する。

- ワークステーション 「パッケージグループの選択」で「ソフトウェア開発」を追加する。
- サーバ 特になし。
- ラップトップ 「パッケージグループの選択」で「ソフトウェア開発」を追加する。

- カスタム「パッケージグループの選択」で「ソフトウェア開発」と「ネットワークサポート」か、「個々のパッケージを選択」で「TCPWrapper」を選択する。

また、インストール時に指定するファイルシステムは ext2 および ext3 のどちらでもよい。

2.1.4. SELinux カーネル構築方法

ここでは SELinux カーネル構築方法について説明する。

(1) 追加されるカーネルオプション

LSM パッチを Linux カーネルソースに適用することで、Linux のカーネルオプションに「Security options」の項目が追加される。これは SELinux で提供される機能をカスタマイズする項目である。「Security options」の中には以下に示す選択項目がある。

- Capabilities Support

Linux カーネルで提供されている Linux Capability 機能を有効にする。デフォルトでは「Yes」となっている。

- IP Networking Support

ネットワークキングフックによって IP のネットフィルタ機能を有効にする。デフォルトでは「Yes」となっている。

- NSA SELinux Support

SELinux セキュリティモジュールを有効にする。1つのモジュールとして構築できるが推奨されていない。このオプションを有効にする場合は、「Capabilities Support」と「IP networking hooks」も有効にすべきである。デフォルトでは「Yes」となっている。

- NSA SELinux Development Module

SELinux セキュリティモジュールを開発モジュールとして構築する。開発モジュールでは、セキュリティポリシーで許可されていないアクセスが発生したときにアクセス内容だけを記録し、セキュリティポリシーを施行しない。新たにセキュリティポリシーを設定するときや変更するとき、このオプションが有効になっている SELinux カーネルでシステムを起動することによりセキュリティポリシーの設定を容易にできる。デフォルトでは「Yes」になっている。

- NSA SELinux MLS policy(EXPERIMENTAL)

SELinux セキュリティモジュールに MLS ポリシーコンポーネントを追加する。ただし MLS ポリシーコンポーネントはまだ実験段階である。「Code maturity level options」項目にある「Prompt for development and/or incomplete code/drivers」を有効にしないとこの項目は選択できない。デフォルトでは「No」になっている。

- LSM port of Openwall(EXPERIMENTAL)

Openwall⁷のカーネルパッチを LSM に適用することを有効にする。この Openwall のカーネルパッチは Openwall 公認ではない。

このオプションを有効にすると「Add RLIMITS_NPROC check to execve」、「Restricted links in /tmp」および「Special handling of fd 0, 1, and 2」オプションを設定できるようになる。

上記の MLS のオプションと同様に、「Code maturity level options」項目にある「Prompt for development and/or incomplete code/drivers」を有効にしないとこの項目は選択できない。このオプションはまだ実

験段階なのでデフォルトでは「No」になっている。

- Add RLIMITS_NPROC check to execve

このオプションは Openwall CONFIG_SECURE_RLIMIT_NPROC パッチを LSM に適用する。「LSM port of Openwall」オプションが有効であるときに選択できる。

このパッチを適用することで、execve システムコールにおいて RLIMITS_NPROC (カレントプロセスが生成できる最大のプロセス数) を引数として setrlimit システムコールを実行し、プロセス生成の上限がチェックされるようになる。通常の Linux では、setrlimit システムコールによってユーザが生成できるプロセス数を制限した場合、fork システムコールを実行時にだけチェックされる。プロセスが UID を変更した場合にこの制限を越えてプロセスを生成することができる。UID を変更するには特権が必要となるのでこのことがセキュリティに関する問題ではない。しかし、プロセスのリソース制限を設定してユーザコンテキストに切り替わる特権を持つプログラムがある。UID を切り替える前に fork システムコールが発行されるために、RLIMITS_NPROC に対するチェックが行なわれないことになる。デフォルトでは「No」になっている。

- Restricted links in /tmp

このオプションは Openwall CONFIG_SECURE_LINK パッチを LSM に適用する。「LSM port of Openwall」オプションが有効であるときに選択できる。

このパッチを適用することで、スティッキービット (ファイルプロテ

⁷ [http:// www.openwall.com/](http://www.openwall.com/)

クションモードのひとつ)のモードが設定されているディレクトリへのハードリンクを抑止する機能を有効にする。デフォルトでは「No」になっている。

- Special handling of fd 0, 1 ,and 2

このオプションは Openwall CONFIG_SECURE_FD_0_1_2 パッチを LSM に適用する。「LSM port of Openwall」オプションが有効であるときに選択できる。

このパッチを適用することで、SUID/SGID ビットが設定されたプログラム実行時にファイルディスクリプタ (fd) の 0 番 (標準入力)、1 番 (標準出力)、2 番 (標準エラー出力) がオープンされていることが保証される。fd がクローズされると /dev/null がオープンされて割り当てられる。デフォルトでは「No」になっている。

- Domain and Type Enforcement (EXPERIMENTAL)

DTE を有効にする。DTE についての詳しい情報は URL <http://www.cs.wm.edu/~hallyn/dte/> を参照のこと。

「Code maturity level options」項目にある「Prompt for development and/or incomplete code/drivers」を有効にしなければこの項目は表示されない。このオプションはまだ実験段階なのでデフォルトでは「No」になっている。

(2) LSM カーネルパッチが適用してある Linux カーネルの場合

ここでは、配布形態 1 の LSM パッチがすでに適用してある Linux カーネルを使ったカーネル構築の方法を説明する。

SELinux のホームページよりファイルをダウンロードし、/usr/src などの適切なディレクトリに展開する。ダウンロードしたファイルを展開すると2つのディレクトリ、lsm と selinux が作成される。lsm ディレクトリは LSM パッチが適用してある Linux カーネルソースが入っており、selinux ディレクトリは SELinux のユーティリティなどが入っている。

```
# cd /usr/src
# tar zxvf /ファイルが存在するディレクトリ名/ファイル名
# ls
lsm  selinux
```

まず、LSM パッチが適用してあるカーネルソースに、デフォルトの「Security Option」設定を適用するパッチと、SELinux の構築に必要なオプションを Makefile に反映させるパッチを適用する。

```
# cd selinux/module
# make insert
```

これにより SELinux カーネルを構築する準備が整ったので、通常の Linux カーネル構築と同様に SELinux のカーネル構築を行なう。

最初に、使用しているハードウェアおよび利用したい機能に合わせてカーネルオプションを設定する。そして、SELinux として構築するために「Networking Options」の「Network Packet Filtering」を有効にし、「Security Options」の設定項目を必要に応じて有効にする（先の make insert によって、最低限の SELinux カーネル構築に必要な設定オプションは有効になっている）。また構築先のマシンが SCSI 接続されているハードディスクから Linux が起動している場合は、SCSI カードのデバイスドライ

バをモジュール化してはいけない。必ず使用する SCSI カードのデバイスドライバをカーネルに含めて構築する。

```
# cd ../../lsm
# make menuconfig (または make xconfig)
# make dep
# make
# make modules
```

以上で SELinux のカーネル構築は完了である。

(3) LSM カーネルパッチのみ

ここではすでに Linux カーネルを持っており、SELinux カーネル構築に必要なファイルだけをダウンロードする場合（配布形態 3 のファイル）の構築方法を説明する。

まず SELinux のホームページよりファイルをダウンロードしてくる。また、Linux カーネルが入っているディレクトリ名を linux から lsm に変更する。

```
# cd /usr/src
# mv -i linux lsm
```

すでに展開されている Linux カーネルソースに LSM パッチを適用する。

```
# cd lsm
# patch -p1 <../lsm-selinux.patch
```

次にダウンロードした selinux.tar.gz ファイルを lsm ディレクトリがあるディレクトリに展開する。

```
# cd ..  
# tar zxvf selinux.tar.gz
```

あとは(2)LSM カーネルパッチが適用してある Linux カーネル構築の場合と同様に構築する。

2.1.5. SELinux のインストール

ここでは SELinux のインストールについて説明する。

2.1.5.1. インストールの方法

SELinux のインストール方法には1つ1つコマンドを入力していく通常のインストール方法と、それらのコマンドが記述してあるスクリプトの実行による quick インストールがある。またカーネル構築の際にカーネルオプションの設定で、「Security Option」の「NSA SELinux MLS policy(EXPERIMENTAL)」を有効にした場合は、いくつかのファイルを編集する必要があるので MLS を有効にした場合のインストール方法についても説明する。

(1) 通常のインストール

以下の手順に従ってインストールする。

(A) カーネルインストール

構築した SELinux カーネルをインストールする。

```
% su (スーパーユーザでない場合)  
# cd /usr/src/lsm  
# make install  
# make modules_install  
# cd ..
```

(B) 必要なファイルのインストール

SELinux をインストールする際に必要なプログラムをインストールする。

```
# cd selinux/module
# make install
# cd ..
```

(C) ファイル users の編集

/usr/src/selinux/policy ディレクトリにある users ファイルをログインするユーザごとに設定する。この users ファイルでは、ユーザがなれる role を、ログインするユーザごとに定義する。users ファイルにエントリがないユーザはログインすることができないので注意が必要である。

デフォルトの SELinux ではユーザに付与できる role として次の 2 つが用意されている。

- sysadm_r システム管理者に付与する role。
- user_r 特権を与えない一般ユーザに付与する role。

また users ファイルにはユーザ ID system_u のエントリがある。これは init などのシステムプロセスとオブジェクトのためのユーザ ID であり、UNIX でのユーザ ID に関連付けされることはない。system_u に付与されている role は system_r であり、デフォルトで定義されているすべての domain に遷移できる。各 role で遷移できる domain は rbac ファイルで定義されている。

デフォルトの users ファイルには root ユーザと、例としてダミーユーザのエントリが記述してある。今回は不要なユーザを削除するだけであ

る。これによってインストール後にログインできるユーザは root のみになる。この設定ファイルはインストール後でも設定することができるので、新たにログインさせるユーザを追加するには、2.1.6.7を参照して設定する。

```
# vi policy/user
以下の行を削除
user jadmin roles { user_r sysadm_r };
user jdoe roles { user_r };
```

(D) ポリシー設定ファイルの編集

SELinux で提供されているポリシー設定ファイルを必要に応じて編集する。Xサーバに付与される domain のポリシー設定がまだ不完全なので、デフォルトの設定では X サーバを起動することができない(「NSA SELinux Development Module」が無効になっている場合)。X サーバを起動させるには X サーバの domain に関する設定ファイルを編集する。policy/domains/program/xserver.te ファイルの「#Comment out by default」が記述されている定義文を有効にする。以下の行を有効にする。

```
# vi policy/domains/program/xserver.te
以下の定義文を有効にする。
#allow $1_xserver_t $1_xserver_t:capability
{net_bind_service setuid setgid chown dac_override fsetid sys_rawio
mknod };

#allow $1_xserver_t memory_device_t:chr_file rw_file_perms;

#allow $1_xserver_t memory_device_t:chr_file x_file_perms;
```

また SELinux ではコンソールログインでのマルチユーザモード(RedHat Linux ではランレベル 3) で起動することを推奨している。X ディスプレ

イマネージャを使ったログインをする場合はSELinux用に拡張されたgdmをインストールしなければならない。これは、ログインするときにユーザのroleとtypeを選択する必要があり、一般のXディスプレイマネージャではこれらを選択できないからである。roleとtypeを選択できる拡張されたgdmのソースファイルは、WebサイトSourceForegeのSELinux Project Site⁸よりダウンロードできる。

現在SELinux版のgdmは、RedHat7.2に対応したものと、RedHat6.1からRedHat7.1に対応した2種類しか提供されていない。SELinux版のgdmのインストール方法を以下に示す。展開先は特に指定はない。

通常のgdmを使用しないように名前を変更する。

```
# mv /usr/X11/bin/gdm gdm.org
# cd /usr/src/selinux
# mkdir selinux-gdm
# cd selinux-gdm
# tar zxvf gdm-rh7.2.tgz
# cd gdm-2.2.3.1
# ./configure
# make
# make install
```

Xサーバの設定と同様にデフォルトの設定ファイルではgdmを起動できないので、policy/domains/system/gdm.teファイルの「# Comment out by default」が記述されている定義文を有効にする。以下の行を有効にする。

⁸ URL <http://sourceforeg.net/projects/selinux/>

```
# vi policy/domains/program/gdm.te
以下の定義文を有効にする。
#allow gdm_t self:capability { setgid setuid dac_override
sys_admin sys_rawio fsetid mknod chown kill fowner
ipc_owner };

#allow gdm_t memory_device_t:chr_file { execute read
write };
```

ポリシー設定ファイルは、SELinux で起動しているときにも変更可能である。変更した設定をセキュリティサーバに反映させる方法は2.1.6.11にて説明する。

(E) 日本語環境の設定ファイルについて

デフォルトで用意されている設定ファイルには、日本語環境に特有な canna サーバなどに関する設定ファイルが用意されていない。もし必要があればインストール後、新たに設定ファイルを作成しなければならない。ここでは canna サーバの設定ファイルを追加する。2.1.7.8を参照し、canna サーバのネットワークの設定、TE の設定そして RBAC の設定をする。

(F) ポリシーファイルのインストール

ポリシー設定ファイルの構築とインストールをする。

```
# cd policy
# make
# make install
# cd ..
```

(G) libsecure のインストール

libsecure と libsecure テストプログラムの構築とインストールをする。

```
# cd libsecure
# make
# make install
# cd ..
```

(H) アプリケーションのインストール

SELinux 付属のアプリケーションの構築とインストールをする。

使用している RedHat Linux のバージョンが 7.2 ならば Makefile の「 LOGROTATE_VER=3.5.4-1 」をコメントアウトして「 LOGROTATE_VER=3.5.9」を有効にする。利用している RedHat のバージョンが 7.1 ならば LOGROTATE_VER=3.5.4-1 を有効にする。

```
# cd utils
# vi Makefile
# For RH7.1, use these versions.
#LOGROTATE_VER=3.5.4-1   RadHat7.1 ならば有効にする
# For RH7.2, use these versions.
LOGROTATE_VER=3.5.9   RadHat7.2 ならば有効にする

# make
# make install
# cd ..
```

(I) 設定ファイルのコピー

utils/appconfig ディレクトリにあるすべてのファイルを /etc/security ディレクトリにコピーする。必要に応じて default_context、default_type、cron_context および initrc_context ファイルを編集する。これらのファイルではそれぞれ以下のことを設定

する。

- default_context ファイル

ユーザごとにログインしたときのデフォルトの role と domain を定義する。ログイン時に role と domain を選択しなければ、このファイルで指定した role と domain でログインする。デフォルトの role は users ファイルで定義した role から、デフォルトの domain は、指定した role がなれる domain から選択しなければならない。デフォルトでは管理者ユーザ root に、sysadm_r と、ファイルシステムのマウントやネットワークインタフェースの設定といった管理タスクの実行が許可されている domain である sysadm_t が定義されている。特権を与えない一般ユーザには user_r と、シェルの実行やホームディレクトリにあるファイルの読み書き、実行などが許可される domain である user_t が定義されている。

このファイルに新たなユーザエントリを追加するには「ユーザ名:role:domain」の順番で記述する。例えば、新たに特権を与えない一般ユーザ takeuchi を user_r、user_t で追加するには以下のように記述する。

```
takeuchi:user_r:user_t
```

このファイルは SELinux 用に変更された login プログラムと sshd プログラムが参照する。

- default_type ファイル

各 role のデフォルト domain を定義する。このファイルに role のエントリを追加するには「role:domain」と記述する。domain は記述し

た role がなれる domain を記述しなければならない。例えば新たな role として guest_r を追加して、デフォルトの domain を guest_t として記述したい場合は以下のように記述する。

```
guest_r:guest_t
```

このファイルは、SELinux 用に変更された login プログラムと role を変更する newrole コマンドが参照する。

- cron_context ファイル

SELinux 用に変更された crond が使用するセキュリティコンテキストを定義する。ユーザごとに crond が実行されたときの role と domain を設定する。エントリの書式は default_context と同じである。

デフォルトでは、ユーザ ID system_u に logrotate などのシステムが実行する crond の role と domain を記述している。

- initrc_context ファイル

SELinux で新たに導入された run_init コマンドによって実行を開始する rc スクリプトの security context を定義する。デフォルトではユーザ ID を system_u、role を system_r、そして domain を initrc_t としている。initrc_t は、rc スクリプトの domain であり、各種サービスやデーモンなどを起動させるために多くの権限を与えられている。run_init コマンドについては、2.1.7.1(6)(D)を参照のこと。

これらのファイルは各プログラムが実行時に参照するので、インストール後でも変更可能である。今回は、default_context と cron_context ファイルのダミーエントリ jadmin、jdoe を削除する。

```
# cd utils/appconfig
# vi default_context cron_context
削除
  jadmin roles { user_r sysadm_r };
  jdoe roles { user_r };
# cp * /etc/security/.
```

(J) ファイルの削除

インストール先のマシンに、LSM 版でない SELinux を以前にインストールしていた場合は、LSM 版でない SELinux のバージョンで作成されたセキュリティラベルとファイルの対応表を削除する必要がある。この対応表は、各ファイルシステムのルートディレクトリの「...security」サブディレクトリに作成される。例えば次のように削除する。

```
# rm -rf /...security /boot/...security
```

(K) ラベル付けの設定

各ファイルに付加するセキュリティラベルが記述されている `setfiles/file_contexts` ファイルをインストール先の環境に合わせて変更する。`file_contexts` は RedHat Linux をベースに作成されているので今回はそのまま使用する。`file_contexts` ファイルの書式は、2.1.6.10 を参照のこと。`file_contexts` ファイルのエントリにインストール先のホストに存在しないファイルパスがある場合は、ラベル付けの際に注意が表示されるが正常に終了する。

ポリシーファイルの設定で X サーバおよび gdm を使用できるように設定した場合は、`setfiles/file_contexts` ファイルで以下の行を有効にす

る。

```
# vi setfiles/file_contexts
以下の行を有効にする。
#/var/log/XFree86.*      system_u:object_r:gdms_log_t
#/tmp/.X11-unix(|/.*)   system_u:object_r:gdms_tmp_t
#/tmp/.X0-lock          system_u:object_r:gdms_tmp_t
```

またポリシーファイルの設定で canna サーバの設定ファイルを追加した場合は、2.1.7.8(4)を参照して canna サーバで使用するファイルのラベル付けのエントリを追加する。

ラベル付けの変更はインストール後でも可能である。再ラベル付けの方法は2.1.6.10を参照のこと。

(L) ラベル付け

SELinux で新たに導入されたセキュリティコンテキストをファイルと対応付けるプログラム setfiles をコンパイルして、file_contexts ファイルを使って setfiles プログラムを実行する。

```
# cd setfiles
# make
# make relabel
# cd ..
```

(M) ブートマネージャの設定

インストールした SELinux カーネルをブートさせるためにブートマネージャの設定を行なう。ここでは LILO と GRUB の場合について説明する。

- LILO を使用している場合

作成した SELinux カーネルのエントリを/etc/lilo.conf ファイルに追加して/sbin/lilo コマンドを実行する。作成した SELinux カーネル名は名前の最後が「-selinux」となっている。また SELinux では、ブート時に SCSI モジュールをロードすることができないので、SELinuxカーネルのエントリを追加する際に SELinuxカーネルのエントリ内に「initrd=」オプションを記述してはならない。

- GRUB を使用している場合

インストール先のホストが RedHat7.2 を使用している場合は、RedHat が提供する「/sbin/new-kernel-pkg」スクリプトを使用することができる。これがない場合「/sbin/grubby」を使用する。

RedHat Linux で提供されている/sbin/new-kernel-pkg スクリプトを使用する場合は以下のように実行する。

```
# /sbin/new-kernel-pkg-install 2.4.16-selinux
```

/sbin/new-kernel-pkg スクリプトでは、新しいカーネルをデフォルトとして起動させるためのオプション設定が用意されていないので grubby コマンドで設定する必要がある。

/sbin/grubby コマンドを使用する場合は以下のとおりである。

```
# /sbin/grubby
  -add-kernel=/boot/vmlinuz-2.4.16-selinux
  -copy-default -make-default -title "SELinux"
```

(N) リブート

システムをリブートして SELinux カーネルで起動する。

(O) SELinux へのログイン

システム管理者としてログインするために role が sysadm_r、domian が sysadm_t であるユーザでログインする。これに該当するユーザは、デフォルト設定では root ユーザである。ここでは root ユーザでログインして sysadm_r と sysadm_t を選択する。

```
Localhost:root
Passwd:****
Your default security context is root:sysadm_r:sysadm_t
Do you want to enter a new security context?[n] ↵
```

(P) 再ラベル付け

SELinux でないカーネルが起動していた場合は再びラベル付けをする必要がある。

```
# cd setfiles
# make
# make verbose
# cd ..
```

(Q) パスの追加

/usr/local/selinux/bin にインストールされた SELinux 用に変更されたユーティリティのパスを追加する。

```
# vi ~/.bashrc
以下を追加する。
PATH=/usr/local/selinux/bin:$PATH
#source ~/.bashrc
#which ls
/usr/local/selinux/bin/ls
```

(R) 動作確認

SELinux の動作を確認するために SELinux 用に変更された ls コマンドでファイルのセキュリティコンテキストを、ps コマンドでプロセスのセキュリティコンテキストを表示させてみる。ファイルおよびプロセスのセキュリティコンテキストを表示させるには ls、ps コマンドとも「--context」をオプションとして付ける。セキュリティコンテキストは「ユーザ ID:role:type:MLS のレベル」となる。MLS のレベルは MLS が有効の場合のみ表示される。

```
# ls --context
drwxr-xr-x root    root    system_u:object_r:bin_t:unclassified bin
drwxr-xr-x root    root    system_u:object_r:boot_t:unclassified boot
drwxr-xr-x root    root    system_u:object_r:device_t:unclassified dev

# ps --context
1229    260 yamada:user_r:user_t:unclassified    bash
1628    260 yamada:user_r:user_t:unclassified    ps -context
```

もし、システムプロセスの domain が initrc_t を付与されているプロセスであれば、まだ設定ファイルによって独立した domain に分割されていないのか、file_contexts で正しくパスが設定されていないかのどちらかである。いかなるプロセスも initrc_t で実行させるべきでないので、必要がなければプロセスを起動させないようにするか、新たに domain を定義する必要がある。

(S) NSA SELinux Development Module の説明

SELinux のカーネルオプション「NSA SELinux Development Module」を有効にして構築した場合は、SELinux モジュールが「permissive mode」で初期化される。「permissive mode」とは、セキュリティポリシーで許

可されていないアクセスが発生したときにそのアクセス内容を記録し、セキュリティポリシーを施行しない状態である。これに対して、セキュリティポリシーを強制施行する状態を「enforcing mode」と呼んでいる。SELinux の状態を「enforcing mode」または「permissive mode」にするには、SELinux で導入された `avc_toggle` プログラムによって変更できる。`avc_toggle` プログラムを実行するには、実行ユーザが管理者用 role である `sysadm_r` と管理者用 domain である `sysadm_t` でなければならない設定となっている。

また「NSA SELinux Development Module」を無効にして構築された SELinux モジュールでは、`avc_toggle` プログラムによって変更することができないので常に「enforcing mode」となる。

(T) ポリシー設定の変更について

上記のインストールではデフォルトで提供されているポリシー設定をそのまま使っているだけなので、必要に応じてポリシー設定を変更する必要がある。ポリシー設定を変更した場合は、ポリシー設定をコンパイルして SELinux モジュールにロードする必要がある。「enforcing mode」で動作しているときは、管理者用 role の `sysadm_r` と管理者用 domain の `sysadm_t` であるユーザでなければならない。

```
# cd /usr/src/selinux/policy
# ポリシー設定ファイルの編集
# make load
```

またファイルのセキュリティコンテキストを変更した場合は、`file_contexts` ファイルを編集してファイルの再ラベル付けをする。

```
# cd /usr/src/selinux/setfiles  
# file_contexts ファイルの編集  
# make relabel
```

SELinux で導入された `chcon` コマンドによってファイルのセキュリティコンテキストを変更することも可能である。`chcon` の使い方は2.1.6.10を参照のこと。

(U) 最後に

SELinux でないカーネルを起動した場合は、SELinux をブートする前にファイルのセキュリティコンテキストをリセットするため `setfiles` プログラムを実行したほうがよい。

(2) quick インストール

SELinux で提供されている `quick` インストールは、カーネルの構築および通常のインストールで入力したコマンドを自動的に実行していくスクリプトである。`quick` インストールで SELinux をインストールする場合は以下の手順で行なう。説明ではカレントディレクトリを/`ファイル展開先ディレクトリ/selinux`としている。

(A) ファイルの展開

カーネル構築および、SELinux インストールに必要なファイルを展開しておく。

(B) users ファイルの設定

通常インストールと同様に `policy` ディレクトリにある `users` ファイルを設定する。

(C) セキュリティコンテキストの編集

utils/appconfig ディレクトリにある default_context ファイルと cron_context ファイルを編集する。これらのファイルの説明は 2.1.5.1(1)(I)を参照のこと。

(D) ラベル付けの設定

通常インストール(1)(K)と同様に setfiles/file_contexts ファイルをインストール先のホスト環境に合わせて変更する。

(E) ポリシー設定ファイルの編集

通常インストール(1)(D)と同様に必要に応じて X サーバの設定とランレベルの設定を行なう。

(F) Makefile の編集

インストール先の Linux が RedHat Linux7.2 ならば、utils ディレクトリにある Makefile の LOGROTATE_VAR 定義を通常インストール(1)(H)と同様に変更する。

(G) インストール開始

quick インストールを開始する。

```
# su
# cd /usr/src/selinux
# make quickinstall
```

(H) インストール後の設定

通常インストール(1)(M)からインストールを続ける。

(3) MLS を有効にする場合

カーネル構築の際に、MLS ポリシーを SELinux に組み込むオプション「NSA SELinux MLS policy(EXPERIMENTAL)」を有効にする場合は、カーネル構築およびインストールの際に以下のことをする。

(A) カーネルオプション設定

SELinux カーネルを構築する際のカーネルオプションの設定で、「Code maturity level options」項目にある「Prompt for development and/or incomplete code/drivers」を有効にする。この項目を有効にしないと「Security Options」項目内で「NSA SELinux MLS policy(EXPERIMENTAL)」が選択できない。そして「NSA SELinux MLS policy(EXPERIMENTAL)」を有効にする。

(B) Makefile の編集

通常のインストール手順(1)(B)で module ディレクトリにおいて「make install」を実行する前に、module/checkpolicy ディレクトリにある Makefile を以下のように編集する。これは checkpolicy プログラムが MLS ポリシー設定を解釈できるようにするためである。

```
# cd/ usr/src/selinux/module/checkpolicy
# vi Makefile
MLS=n
    このように変更する
MLS=y
# make install
```

(C) ユーザの MLS 範囲の設定

通常インストール(2)(B)で、policy ディレクトリにある user ファイル

に各ユーザがアクセスできる MLS の範囲を記述する。書式は以下である。

```
ユーザ名:role:type  
[:sensitivity[:category,...]][-sensitivity[:category,...]]
```

MLS の範囲は、sensitivity に機密情報を、category にカテゴリを記述する。カテゴリは複数指定できる。上記の左側にある[:sensitivity[:category,...]]が最下限ラベルとなり、右側が最上限ラベルとなる。最上限ラベルがなければ最上限ラベルが最下限ラベルと同じになる。また、MLS 範囲の記述がなければデフォルトで、「unclassified」となる。MLS については、2.1.6.6で詳しく説明する。

(D) Makefile の編集

通常インストール手順(1)(F)でポリシー設定を構築する際に policy ディレクトリにある Makefile を以下のように編集する。

```
# cd /usr/src/selinux/policy  
# vi Makefile  
MLS=n  
    このように変更する  
MLS=y  
# make install
```

2.1.6. SELinux の設定方法

ここでは SELinux のセキュリティポリシー設定方法について説明する。SELinux のセキュリティポリシー設定は、SELinux で実装されている TE、RBAC、MLS の設定である。MLS の設定はカーネル設定オプションで MLS を有効にした場合だけである。

SELinux は、実装されているアクセス制御で許可されたアクセスならば、通常

の Linux で実装されているファイルパーミッションによるアクセス制御チェックも行なう。設定をさらに強固なものにするためにも SELinux のポリシー設定だけでなく、ファイルパーミッションの設定も厳密に行ったほうがよい。

2.1.6.1. 設定の概要

セキュリティポリシー設定は、最初に TE の domain と type を宣言する。プロセスには domain が付与され、オブジェクトには type が付与される。ポリシー設定で、domain と type 間にアクセスベクタと呼ばれるパーミッションを定義する。type が付与されるオブジェクトには、ファイル、ディレクトリ、ソケットなど 29 種類ある。アクセスベクタはオブジェクトの種類ごとに異なっている。さらに domain 間のシグナル送信や Linux Capability も制限できる。これらを定義することで細かいアクセス制御を行なうことができる。そして、domain がプログラムファイルに付与された type を実行したときにどの domain に遷移するかといった domain 間で許容する遷移を記述する。

また、role を宣言して role によって遷移できる domain 集合を記述する。

次にデフォルトで用意されている設定ファイルの目的と、その目的の実現方法について説明する。

(1) raw アクセスの制御

/dev/mem や/dev/kmem などのデバイスに raw アクセスすることを制御することである。これらの各種デバイスに対して別々の type を定義する。そして、定義した type にアクセスする必要があるプロセスに別々の domain を定義する。

(2) カーネルの完全性

ポリシー設定ではカーネルの完全性を実現するためにブートファイル、

モジュールオブジェクトファイル、モジュールユーティリティ、モジュール設定ファイルそして `sysctl` パラメータを別々の `type` で定義している。そして、これらのファイルへの書き込み権限が必要なプロセスを別々の `domain` として定義する。

(3) システムプログラム、設定情報とログの完全性

ポリシー設定ではシステムライブラリとシステムバイナリへのアクセスを制限するため、これらのファイルに別々の `type` を定義する。そしてシステムソフトウェアの変更は管理者にしか許可させない。設定情報とログに別々の `type` を定義して書き込み権限が必要なプログラムに対して別々の `domain` を定義する。

(4) プログラムの封じ込め

攻撃者が特権を持つプロセスを利用して与える被害を封じ込めることである。ポリシー設定では、必要最低限のアクセス権限のみ与えられた `domain` で特権を持つプロセスやプログラムを実行させる。

(5) 悪意あるコードからの保護

悪意あるコード実行から特権を持つプロセスを保護することである。ポリシー設定では特権を持つプロセスが実行するプログラムに `type` を定義し、その `type` を実行することによって特権を持つ `domain` に遷移することを許可する。管理者 `domain` がシステムソフトウェアのような管理者によって起動されるプログラムの実行を許可される。

(6) 管理者権限の保護

ユーザ認証なしで入ったユーザから管理者の `role` と `domain` を保護することである。ポリシー設定では、`login` プログラムによってのみ管理者の

role と domain になることを許可している。リモートログインは、.rhosts ファイルによって認証なしでログインできることから管理者の role と domain になることを制限する。

(7) システムプロセスや管理者プロセスの保護

一般ユーザのプロセスがシステムプロセスや管理者のプロセスを勝手に操作することを制限する。ポリシー設定では、異なる domain のプロセスエントリ procfs へのアクセスを信頼できるシステムプロセスと管理者だけに許可している。これは ptrace コマンドの使用と domain 間でのシグナル送信を制御できる。

(8) 悪意あるモバイルコードからのユーザ保護

ブラウザの欠陥を利用する悪意あるモバイルコードからユーザと管理者を保護することである。ポリシー設定ではブラウザの domain を定義してアクセスを制限する。

2.1.6.2. マクロ定義

SELinux の設定ではマクロを使用することができる。マクロは、m4 マクロプロセッサによって展開される。マクロを使用することで設定ファイルの記述を簡単に行なえる。例えばファイルの読み込みに必要なパーミッションは、read、getattr、access、lock、poll、ioctl である。これらをマクロ名 r_file_perms としてマクロ定義することで、それぞれを記述する手間が省ける。また、ファイルの読み込みに必要なパーミッションをすべて知らなくともマクロ名を知っていれば容易に設定ができる。

マクロの定義方法は以下のとおりである。

```
define( `マクロ名` , `置換内容` )
```

マクロを呼び出す場合は、単にマクロ名を記述すれば定義した置換内容と置き換わる。m4 マクロでは、マクロを呼び出すときに引数を渡すこともできる。これには、定義するマクロの置換内容において n 番目の引数を表す \$n 記号を使用する。例えば、\$1 は一番の引数と置換される。マクロの呼び出しは TE と RBAC の設定ファイルにおいて以下の書式で記述する。

```
マクロ名[ (引数 1, 引数 2, ...)]
```

マクロを呼び出す際に引数が少なければ空文字で置換される。引数が多いときは無視される。

macro.te ファイルであらかじめいくつかのマクロが定義されている。付録3.1 に macro.te で定義されているマクロの一覧がある。

2.1.6.3. 設定ファイル一覧

SELinux のポリシー設定に必要なファイルは、デフォルトで /usr/local/selinux/flask ディレクトリと /展開先のディレクトリ /selinux/policy にある。ここでは各ファイルの内容について説明する。

(1) /usr/local/selinux/flask ディレクトリ

次の3つのファイルが存在する。これらのファイルは特定のセキュリティポリシーと関係なく、通常は管理者が設定を行なうことはない。

- security_classes ファイル

セキュリティオブジェクトのクラス一覧が記述されているファイル。各

クラスに対応する定数がカーネル構築時に自動作成される
/usr/local/selinux/flask/flask.h に定義してある。

- initial_sids ファイル

システム初期処理のためにあらかじめ定義された SID が記述されている
ファイル。対応する定数はカーネル構築時に自動作成される
/usr/local/selinux/flask/flask.h に定義してある。

- access_vectors ファイル

各セキュリティクラスのパーミッション（アクセスベクタ）が定義され
ているファイル。各パーミッションに対応する定数はカーネル構築時に
自動作成される include/linux/flask/av_permissions.h に定義してある。

(2) policy ディレクトリ

管理者はこのディレクトリにあるファイルを編集あるいは、新規に作成
することでセキュリティポリシーを設定する。

まず、TE 設定に関するファイルを示す。

表 2.1-2 TE 設定ファイル一覧

ファイル名	ファイルの説明
all.te	TE に関する設定ファイルを一つに結合したファイルであり、設定ファイルのコンパイル時に自動的に作成される。
assert.te	TE の設定に対するアサーションを定義するファイル。
domains	domain の定義とルールが格納されているディレクトリ。
init.te	システムブート時の TE に関する設定が記述されているファイル。
types	特定の domain に関連付けされないファイルに対して付与する type を定義するファイルを格納するディレクトリ。

domains ディレクトリと types ディレクトリにあるファイル一覧を以下
に示す。

表 2.1-3 domains ディレクトリの一覧

ファイル名またはディレクトリ名	ファイルの説明
admin	システム管理者用の domain の定義が記述されているファイルが格納されるディレクトリ。システム管理者 domain の設定が記述してある sysadm.te ファイルがある。
every.te	すべての domain に適用される domain 属性の定義が記述してあるファイル。
system	デーモンなどのシステムプログラムに関する domain 設定ファイルが格納されているディレクトリ。
program	ユーザが実行するプログラムの domain 設定ファイルが格納されているディレクトリ。
user	一般ユーザ用の domain 定義が記述されているファイルが格納されるディレクトリ。一般ユーザの domain が記述されている user.te ファイルがある。

表 2.1-4 types ディレクトリの一覧

ファイル名またはディレクトリ名	ファイルの説明
device.te	デバイスファイルに付加される type が宣言されたファイル。
devpts.te	devpts に付加される type が宣言されたファイル。
file.te	ファイルに付加される type が宣言されたファイル。
network.te	ネットワークオブジェクトに付加される type が宣言されたファイル。
nfs.te	NFS で使用する type が宣言されたファイル。
procfs.te	procfs に付加される type が宣言されたファイル。
security.te	SELinux で使用するファイルに付加される type が宣言されたファイル。

次に RBAC 設定に関するファイルを示す。

表 2.1-5 RBAC の設定ファイル一覧

ファイル名	ファイルの説明
rbac	RBAC の設定が記述されたファイル。
users	ユーザがなれる role を定義するファイル。 MLS を有効にしている場合は、ユーザがアクセスできる

	MLS の範囲も定義する。
--	---------------

MLS 設定に関するファイルを以下に示す。

表 2.1-6 MLS の設定ファイル

ファイル名	ファイルの説明
mls	MLS の設定を記述するファイル。

セキュリティコンテキストの設定に関するファイルを以下に示す。

表 2.1-7セキュリティコンテキストの設定ファイル一覧

ファイル名	ファイルの説明
fs_contexts	ラベル付けされていないファイルシステムがマウントされたときのセキュリティコンテキストの設定が記述されているファイル。
initial_sid_contexts	初期化 SID のセキュリティコンテキストを定義したファイル。
devfs_contexts	devfs のエントリに対するセキュリティコンテキストを定義するファイル。
net_contexts	ネットワークオブジェクトに付加するセキュリティコンテキストを定義したファイル。

その他の設定に関するファイルを以下に示す。

表 2.1-8 その他の設定ファイル

ファイル名	ファイルの説明
Makefile	ポリシー設定のコンパイルなどを行なうための Makefile。
macros.te	すべての設定で使用するマクロが定義されているファイル。
constraints	セキュリティクラスのパーミッション制約を定義するファイル。
policy	policy.conf をコンパイルしたバイナリファイル。
policy.conf	ポリシー設定に関するすべてのファイルを結合し、マクロ展開したファイル。

2.1.6.4. TE(Type Enforcement)の設定

ここでは TE によるアクセス制御の設定方法について説明する。

(1) 概要

TE の設定は以下の 7 つの文を使って記述していく。

- type の宣言
- アクセスベクタルール
- type 遷移ルール
- type メンバルール
- type の変更ルール
- type の複製ルール
- アクセスベクタアサーション

次にこれらの記述を使った TE の設定方法を説明する。

(2) type の宣言

(A) type の宣言

TE の設定では、まずサブジェクトとオブジェクトに付加する type を宣言する。type 宣言の書式は以下のとおりである。

```
type type 名 [alias {エイリアス 1 エイリアス 2 ...}]  
                [,属性 1,属性 2,...];
```

type 文ではオプションとして type 名のエイリアスと type の属性を加えることができる。エイリアスを指定すれば TE 設定の記述中で type 名の代わりにエイリアス名を使用することができる。エイリアスを 1 つしか宣言しなければエイリアス名を中括弧で括る必要はない。属性は同じ

属性を持つ type が共通した特性を持つことを示すために使用される。属性が後述するルールの中で使われていると、ルールがその属性を持つすべての type に適用される。例えばある type の読み込みを許可するルール中で type 名の代わりに属性を使うと、その属性を持つ type も同様の許可を持つことになる。

SELinux ではサブジェクトに付加する domain とオブジェクトに付加する type を 1 つの抽象的な type として扱っている。以下のように type 名だけの宣言をすると domain とオブジェクト type の両方で使用することができる。

```
type guest_t;
```

ここで、guest_t を domain として使うとなると、domain として必要なアクセス権限を guest_t に対して許可しなければならず、膨大な手間となってしまう。そこで、domain としての必要なアクセス権限が付与される属性 domain を使用する。

```
type guest_t, domain;
```

デフォルトでは属性 domain 以外にも実行ファイルに必要なアクセス権限を付与する属性 exec_type などが用意してある。

属性を用いると type 文を見ただけで、宣言している type がどのようなアクセス権限を持つのか理解しやすくなる。例えば以下のように定義した type について説明する。ここで属性 privlog は syslogd との通信に必要な許可が付与される。http_t は、domain 属性が付与されていること

からプロセスに付与される type であり、また privlog 属性から syslogd を介してログ出力するプロセスであることが type 文を見ただけで理解できる。

```
type http_t, domain, privlog;
```

(B) 属性

デフォルトで用意されている属性を以下の表に示す。表 2.1-9は domain に付与する属性、表 2.1-10は type に付与する属性、そして表 2.1-11はネットワーク関連のオブジェクトに使用する属性である。

表 2.1-9 domain の属性

属性名	説明
domain	この属性を持つ type を domain として使用することを示す。この属性を付与することで、プロセスに必要な権限を許可する。
prvivuser	ユーザ ID を変更できることを示す。この属性を持つ type を付与されているプロセスしかユーザ ID を変更することができない。例えば、init rc スクリプトに付与される initrc_t が持つ。
privrole	role を変更できることを示す。この属性を持つ type を付与されているプロセスしか role を変更することができない。例えば、login プログラムは、ログインしたユーザによって role を変更しなければならないのでこの属性を持つ。
privowner	オブジェクトに付与されたセキュリティコンテキストを他ユーザであっても変更できることを示す。
privlog	syslogd と UNIX ドメインソケットを使って通信することを示す。syslogd でログを採取する domain に付与される。
privmem	メモリデバイスへアクセスすることを示す。/dev/mem にアクセスして、直接、メモリに書き込みなどをする domain に付与される。klogd や X サーバの domain に付与されている。
auth	ユーザ認証をすることを示す。ユーザ認証を必要とする domain に付与される属性である。デフォルトでは local_login_t などにこの属性がついている。しかし、デフォルトでは auth に対する設定はされていない。
userdomain	ユーザプロセスに付与される domain であることを示す。シェルの実行権限などが与えられる。

属性名	説明
admin	管理者用の domain であることを示す。システムライブラリやシステムコマンドへの書き込みや削除権限が許可される。

表 2.1-10 type の属性

属性名	説明
exec_type	実行可能ファイルであることを示す。
sysadmfile	システム管理者がフルアクセスできるファイルであることを示す。デフォルトでは、sysadm_t domain がシステム管理者の type となる。
file_type	ファイルであることを示す。デフォルトの設定では、sysadm_t が file_type に再ラベル付けが許可されている。
fs_type	ファイルシステムであることを示す。ブート時にファイルシステムに対するアクセス権をすべて与える。
pidfile	/var/run ディレクトリ以下に作られる*.pid ファイルであることを示す。domain に対して読み込みの許可を与える。
tmpfile	/tmp ディレクトリなどに作成されるテンポラリファイルであることを示す。
home_type	ユーザのホームディレクトリであることを示す。domain に対して、読み込み権限を許可する。
root_dir_type	root のディレクトリであることを示す。まだ定義されていない。

表 2.1-11 ネットワークオブジェクトの属性

属性名	説明
netif_type	ネットワークインタフェースであることを示す。
netmsg_type	ソケットが受信するメッセージを示す。
node_type	ノードであることを示す。
socket_type	ソケットであることを示す。まだ定義されていない。
port_type	ポートであることを示す。まだ定義されていない。

(3) アクセスベクタールール

アクセスベクタールールでは、domain と type 間および domain 間で許可するパーミッションを記述する。またアクセスが許可された場合にログを出

力するパーミッションと、アクセスが拒否された場合にログを出力するパーミッションも記述する。ログは/var/log/messages ファイルに出力される。以下にアクセスベクトルルールを定義する際の書式を示す。

```
(allow|auditallow|auditdeny|notify)  
ソース type ターゲット type:class 名 パーミッション ;
```

各アクセスベクトルルールの使い方は以下のとおりである。

- allow アクセスを許可するアクセスベクトルルールを示す。
- auditallow アクセスが許可された場合にログを出力するアクセスベクトルルールを示す。
- auditdeny アクセスが拒否された場合にログを出力するアクセスベクトルルールを示す。実際には、アクセスが拒否された場合にログを出力しないパーミッションを記述するために使われる。
- notify セキュリティサーバに指定したパーミッションが使用されたことを知らせる。まだ使用することはできない。

ソース type はアクセスをする domain であり、ターゲット type はアクセスされる domain または type となる。class 名は、ターゲット type のクラスとなる。パーミッションは class 名で指定された class のパーミッションとなる。また、ソース type とターゲット type のところに属性名を記述するとその属性を持つ type にルールが展開される。つまり、その属性を持つすべての type についてアクセスベクトルルールを記述したことと同じである。

アクセスベクトルルールによってログの出力について定義しなければ、アクセスが拒否された場合のみ拒否されたアクセス内容が出力される。アク

セスが拒否された場合にログを出力しないパーミッションを定義するには、`auditdeny` を使ってパーミッションを記述するところで、`~{ログを出力しないパーミッション}`と記述する。以下にアクセスベクトルールの例を示す。

```
allow user_t shell_exec_t:file
    { read getattr access lock poll execute ioctl };
allow domain tmp_t:dir rw_dir_perms;
allow user_t file_type:file r_perm;
auditallow { initrc_t admin } kernel_t:system avc_toggle;
auditdeny domain ld_so_cache_t:file ~write;
```

一番目の例では、一般ユーザのプロセスに付与される `user_t` にシェルの実行ファイルに付与される `shell_exec_t` に対する実行権限を与えている。2番目は、属性 `domain` が `tmp_t` を付与されたディレクトリへの読み書き権限を与えている。3番目は、`user_t` が属性 `file_type` に対して読み込み権限を与える。4番目は、`initrc_t` が属性 `admin` が付与されている `domain` が `avc_toggle` コマンドを実行した場合にパーミッション `avc_toggle` を使用したことをログとして出力する。5番目は、属性 `domain` を付与された `domain` が共有ライブラリキャッシュに付与される type `ld_so_cache_t` に書き込みしようとしたときに、アクセスが拒否されてもログを出力しない。

アクセスベクトルルールでターゲット type を「self」と記述すると、ソース type と同じ type を記述したことになる。つまり、ソース type 自身へのアクセスベクトルルールを定義したことになる。また、ルールの中で、すべてを表すアスタリスク「*」と、補集合を表すチルダ「~」を使用することができる。例えば、ターゲット type のところにアスタリスクを記述すると、宣言されているすべての type を記述したことになる。パーミッションのところでは、class 名にあるすべてのパーミッションを記述したことになる。

る。また、ソース type のところに、「~etc_type」と記述したら、etc_type 以外のすべての type を記述したことになる。例えば以下のように記述する。

```
allow domain self:process *;
```

上記の例では、属性 domain が付与されている domain が、自身にクラス process のすべてのアクセスベクタを許可していることになる。

(4) type 遷移の設定

type の遷移ルール記述では新たに生成されるプロセス、または、新たに作成されるファイルに付与する type を定義する。type の遷移ルールの設定は type_transition 文を使う。以下に書式を示す。

```
type_transition サブジェクト type オブジェクト type  
:class 名 遷移後の type;
```

サブジェクト type はプロセスまたはファイルを作成するサブジェクトに付加されている type である。オブジェクト type は、生成されるプロセスまたは作成されるファイルに関するオブジェクトに付加されている type である。プロセスの生成ならば、関係するオブジェクトは実行ファイルであり、ファイルの作成ならば作成先のディレクトリとなる。class 名はオブジェクト type のクラスを記述する。type の遷移ルールを定義しなければ、プロセスを新しく生成する場合は親プロセスと同じ type である。また、プロセスが新しく作成するファイルの type は作成先のディレクトリと同じ type となる。

サブジェクト type とオブジェクト type は複数の type を記述することで

まとめて定義できる。また、サブジェクト type とオブジェクト type のところに属性名も記述できる。属性名を記述すると属性を持つ type に適用される。アスタリスクを使用した場合すべての type を表すことになるため、定義されているすべての type に対して、定義したルールを付与することになるので注意が必要である。サブジェクト type、オブジェクト type と class 名が同じで、遷移先の type が異なるルールが記述されていると、ポリシーコンパイラが注意を表示して最後に記述されたルールが適用される。以下に type の遷移ルール例を示す。

```
type_transition initrc_t httpd_exec_t:process httpd_t;  
type_transition user_t tmp_t:dir user_file_t;
```

1 番目の例は、initrc_t を付加されたプロセスが httpd_exec_t を付加されたファイルを実行すると、新たに生成されたプロセスに httpd_t を付加する。2 番目の例は、user_t を付加されたプロセスが tmp_t を付与されたディレクトリにファイルを作成すると、ファイルに user_file_t を付与する。

type_transition 文では、遷移を許可しただけで遷移するときに必要なアクセス権限を別に記述しなければならない。例えば、遷移先の domain に対してプロセスクラスのパーミッション transition など許可(allow initrc_t httpd_t:process transition)しなければならない。そこで実際の設定ではマクロを使用して、type_transition 文の記述と必要となるアクセス権を記述する。domain の遷移を許可させるマクロは domain_auto_trans マクロであり、type の遷移を許可させるマクロは file_type_auto_trans である。これらのマクロの使用方法は以下のとおりである。

```
domain_auto_trans(サブジェクト type ,オブジェクト type,  
                  遷移後の type)  
file_type_auto_trans(サブジェクト type ,オブジェクト type,  
                    遷移後の type)
```

マクロに渡す引数は type_transition 文で指定したものと同一である。
ここで注意することは文の最後にセミコロンを記述する必要がないことである。以下に例を示す。

```
domain_auto_trans(initrc_t,sendmail_exec_t,sendmail_t)  
file_type_auto_trans(user_t,tmp_t,user_tmp_t)
```

domain 遷移マクロの例では、initrc_t ドメインの init プロセスが sendmail_exec_t タイプの sendmail 実行ファイルを起動した時にその sendmail プロセスの domain を sendmail_t に遷移させる。ファイルの type 遷移マクロの例では user_t が tmp_t を付与されたディレクトリで新しくファイルを作成すると、作成されたファイルの type を user_tmp_t に遷移させる。

また、type 遷移に関連するマクロとして domain_trans マクロと file_type_trans マクロがある。このマクロは type 遷移で必要となるアクセス権だけを許可するマクロである。domain_trans マクロは、login プログラムのような同じプログラムを実行した場合に複数の domain に遷移する場合に使う。login プログラムではどの domain に遷移するかは login プログラム実行時には決まっておらず、login プログラム内部でどの domain に遷移するか決定する。このような場合、domain_trans マクロを使って遷移に必要なアクセス権だけを許可する。file_type_trans マクロも、プロセ

スがファイルを出力することは分かっているが、どの type のファイルを作成するかは実行時にならないと分からない場合に使う。

また、type の遷移ルールを設定する際に注意すべき点がある。それは、遷移ルール設定のほかに、RBAC の設定でも許可しておかなければならないことである。サブジェクトには role も付加されているので、付加されている role が遷移する type になることを許可していなければ遷移は許可されない。

(5) type メンバールール

type メンバールールでは、プロセスが polyinstantiated オブジェクトにアクセスする際にどの type に対してのアクセスなのかを記述する。polyinstantiated オブジェクトは、名前が同じでも異なる type が付与されていれば別のファイルとして存在することができる。例えば、root がログインする際に一般ユーザの domain user_t としてログインした場合、root のシェルの設定ファイルなどを編集できなくなる。そこで、polyinstantiated オブジェクトを使用して、root のホームディレクトリにある設定ファイルを root がログインする ドメイン の各々について用意しておけば良い。user_t ドメインでログインした場合には user_t ドメイン用の設定ファイルのみが存在しているように見え、それを編集することができる。type メンバールールの書式を以下に示す。

```
type_member サブジェクト type    オブジェクト type
                                     :class 名  遷移後の type;
```

書式の内容は type 遷移の書式と同じである。現在、まだ polyinstantiated オブジェクトがサポートされていないので、type メンバールールを定義して

も適用されることはない。

(6) type の変更ルール

type の変更ルールの記述はすでにオブジェクトに付加されている type を変更する際のルールを定義する。type の変更ルールの定義は、type_change 文を使用し、domain と変更前のオブジェクト type によって、変更後のオブジェクト type を指定する。書式は文の先頭が type_change になること以外、type の遷移ルールと同様である。以下に書式を示す。

```
type_change ソース type ターゲット type  
:class 再ラベル後の type ;
```

例を以下に示す。

```
type_change user_t tty_device_t:chr_file user_tty_device_t;
```

この例では、ユーザがログインしたときに user_t が擬似端末 tty に付与される tty_device_t を user_tty_device_t に変更する。

(7) type の複製ルール

type の複製とは、ある domain の type 遷移ルールとアクセスベクトルルールを別の domain に複製することである。type の複製ルールは clone 文を使って定義される。clone 文の書式は以下である。

```
clone 複製元の type 複製先の type ;
```

ただし、clone 文では以下の条件に合うルールは複製されない。

- プロセスの type 遷移ルールにおいて遷移先の type 名が複製元または複製先の type である。
- アクセスベクタルールにおいてターゲット type が複製元または複製先の type である。

例えば以下の文により、webadmin_t domain に user_t domain がもつ type 遷移ルールと access vector ルールが複製される。

```
clone user_t webadmin_t;
```

(8) アクセスベクタアサーション

アクセスベクタアサーションの記述ではアクセスベクタルールで決して許可しないパーミッションを指定する。アクセスベクタアサーションの記述は neverallow 文を使用し、書式はアクセスベクタルールと同様である。以下に書式を示す。

```
neverallow ソース type ターゲット type:class 名 パーミッション ;
```

アクセスベクタアサーションの設定で指定されたパーミッションがアクセスベクタルールの中にある場合は、ポリシーコンパイラによってエラーが返される。アクセスベクタアサーションの例としては以下のものがある。

```
neverallow ~admin { lib_t bin_t sbin_t }  
                :file { write append unlink rename };
```

この例では属性 admin を付与されていない type は、システムライブラリ、実行ファイル、システム実行ファイルの変更を許可させない設定になる。

デフォルトでは `assert.te` ファイルでアクセスベクタアサーションを記述している。

(9) 属性の定義

`type` の宣言で説明したように `type` の宣言では、属性を関連付けることで属性が持つルールを、定義する `type` に付与できる。例えば、属性 `privlog` は `domain/system/syslogd.te` ファイルで以下のように定義される。

```
allow privlog devlog_t:sock_file rw_file_perms;  
can_unix_send(privlog,syslogd_t)  
can_unix_connect(privlog,syslogd_t)
```

この記述は `syslogd` デーモンと UNIX ドメインによる通信を許可する。上記の `can_unix_send` と `can_unix_connect` はマクロであり、`syslogd` に対するメッセージの送信と、`syslogd` に接続するのに必要なアクセスベクタルールに置換される。属性を作成するときに `type` 文で宣言する必要はない。この属性を `type` に付与するには `type` 定義で説明したように `type` 文を使って以下のように定義する。

```
type sendmail_t , privlog;
```

これによって、`sendmail_t` に `privlog` の設定が付与される。すなわち、`sendmail_t` に対して以下の設定をしたことと同じことになる。

```
allow sendmail_t devlog_t:sock_file rw_file_perms;  
can_unix_send(sendmail_t,syslogd_t)  
can_unix_connect(sendmail_t,syslogd_t)
```

属性設定をすることで `sendmail_t` が何の許可を持つかが分かりやすくな

る。例では、sendmail_t が syslog と通信するのが type 文をみただけで分かる。

マクロを定義することで属性の機能と同じことを実現できるが、分かりやすさの点からマクロにするのではなく属性設定にすることも検討するとよい。

2.1.6.5. RBAC の設定

RBAC の設定によってユーザごとに権限分割ができるようにする。

RBAC の設定では role の宣言、role 遷移ルールの設定、role 遷移許可ルールの設定と role dominance 定義をする。また、role はプロセスだけに関連するのでオブジェクトにはあらかじめ定義してある object_r が付与される。オブジェクトに付与される role は、アクセス制御では使用されないため object_r に関する role 遷移は決して定義されることはない。デフォルトでは rbac ファイルで設定を記述している。ここでは、これらの記述を使って RBAC の設定方法を説明する。

(1) role の宣言

role の宣言では宣言する role 名と、その role が付与されたプロセスが遷移できる type の集合を定義する。role の宣言は role 文を使って定義する。以下に書式を示す。

```
role ロール名 types type 名;  
role ロール名 types { type 名 1 type 名 2...};
```

複数の type を記述する場合は空白で区切って記述する。この type 宣言によってプロセスが遷移できる type を限定することができる。以下に例を

示す。

```
role user_r types user_t;  
role sysadm_r types { sysadm_t ifconfig_t };
```

1 番目の例では、user_r を role として宣言してこの role がなれる type を user_t のみと定義している。2 番目の例では、sysadm_r が遷移できる type を sysadm_t と ifconfig_t であると定義している。

新たに role を宣言した場合にその role でログインするユーザがいるときには、/etc/security/default_type にデフォルトの type を定義しなければならない。

(2) role 遷移許可ルール

role 遷移許可ルールの記述では、ある role が別の role に遷移することを許可する。この記述がなければ role の遷移は決して許可されない。role 遷移許可ルールは allow 文を使って記述する。以下に書式を示す。

```
allow  遷移元 role  遷移先 role;
```

以下に例を示す。

```
allow sysadm_r  user_r;
```

上記の例では、管理者用 role である sysadm_r が一般ユーザ用 role である user_r に遷移することを許可する。

(3) role の遷移ルール

role 遷移ルールの記述では、type 遷移ルールと同様にあるプログラムの

実行時、新たに生成されるプロセスに付与される role を定義する。以下に書式を示す。

```
role_transition 現在の role オブジェクト type 遷移先の role;
```

上記の type は実行可能なファイルに付与されている type を指定する。デフォルトでは、role 遷移ルールによって定義しなければ親プロセスの role が付与される。

role が別の role に遷移するためには role 遷移ルールの記述だけでなく、role 遷移許可ルールの記述によって遷移を許可されていなければならない。そこで role の遷移を行ないたいときは、rbac ファイルに定義してある role_auto_trans マクロを使う。このマクロを使うことで、role 遷移ルールと必要となる role 遷移許可ルールの定義をまとめてできる。以下に使い方示す。

```
role_auto_trans(現在の role, プログラムの type, 新しい role)
```

role が遷移するプログラムを作ると、role によってユーザ権限を分割している意味がなくなる可能性があるので注意が必要である。デフォルトの設定では role_auto_trans マクロは使われていない。以下に例を示す。

```
role_auto_trans(system_r, aa_exec_t, aa_r)
```

aa_exec_t 実行時に role が system_r から aa_r に変わる。

(4) role dominance 定義

role dominance 定義では、宣言されている role 間の階層関係を定義する。階層関係で、上位にいる role は、それよりも下位の role が遷移できる type を自動的に継承する。

まだ role dominance 定義を使用することはできない。

2.1.6.6. MLS の設定

SELinux で実装されている MLS はマルチレベルセキュリティモデルの Bell LaPadula (BLP) を拡張したものである。MLS ではサブジェクトとオブジェクトにレベル範囲をラベル付けする。サブジェクトに付与されているレベル範囲の最下限と最上限が異なっているマルチレベルならば、その範囲内にあるすべてのレベルのデータを扱うことができる。

MLS の設定では機密種別の宣言、機密種別の階層関係の定義、カテゴリの宣言、レベルの定義を行なう。MLS の設定は mls ファイルで記述している。ここではこれらの記述を使って MLS の設定方法を説明する。

(1) 機密種別の宣言

機密種別の宣言ではサブジェクトとオブジェクトに付与する機密種別を記述する。以下に書式を示す。

```
sensitivity 名前 [alias エイリアス名];
```

以下の例では機密種別名が unclassified であり、エイリアスを u としている。その他の例も同様に定義している。

```
sensitivity unclassified alias u;  
sensitivity secret alias s;  
sensitivity top_secret alias ts;
```

(2) 機密種別の階層関係の定義

先に宣言した機密種別の階層関係の定義をする。以下に書式を示す。

```
dominance { 機密種別名 機密種別名 ... }
```

機密種別は先に記述される程、機密種別の機密性が低くなり後ろに記述される程高くなる。以下に例を示す。

```
dominance { u s ts }
```

例ではエイリアスを使って階層関係を定義している。

(3) カテゴリの宣言

情報の分類を表すために使用するカテゴリを宣言する。以下に書式を示す。

```
category 名前 [alias エイリアス名];
```

以下の例ではカテゴリ nato と usuk を宣言している。

```
category nato;  
category usuk;
```

(4) レベルの定義

レベルの定義は、(1)で宣言した各機密種別に、(3)で宣言したカテゴリに関連付けをする。レベルの定義によってオブジェクトにアクセスする際に必要となるカテゴリを定義する。以下に書式を示す。

```
level 機密種別名: カテゴリ名, カテゴリ名, ...;
```

以下に例を示す。

```
level u;  
level s:nato;  
level ts:nato, usuk;
```

例では、機密種別のエイリアスを使ってレベルを定義している。

(5) mls 設定ファイル

mls 設定ファイルでは、アクセスベクタパーミッションと MLS でのパーミッションを対応付けている。これは、MLS の各パーミッションで必要となるアクセスベクタでのパーミッションを示している。

MLS のパーミッションは、read、write、writeby、readby がある。MLS の read パーミッションは、ソース SID の最上限がターゲット SID の最上限よりも優位であるときのみ許可される。MLS の write パーミッションが許可されるのは、ターゲット SID が単一レベルで、かつ、ソース SID の最下限よりも優位のときである。または、ターゲット SID のレベル範囲がソース SID のレベル範囲のサブセットであるときのみである。writeby と readby パーミッションはターゲット SID がソース SID に対してアクセスするときの

パーミッションである。write と read と同様なアクセス条件が必要である。

また MLS のパーミッションが許可されるには、MLS の各パーミッションに対応付けられたアクセスベクタパーミッションも許可されていなければならない。

2.1.6.7. ユーザの設定

SELinux が稼動しているホストにユーザがログインするためには、/etc/passwd ファイルにユーザを追加するだけでなく、SELinux でも設定をしなければならない。

ユーザの設定は users ファイルでユーザのプロセスに付加できる role を定義する。MLS を有効にしているならばユーザがアクセスできる MLS の範囲も定義する。ユーザの設定は user 文を使って定義する。以下に書式を示す。

```
user user 名 roles role_set [ ranges MLS_range_set ];  
role_set := role | {role1,role2,...}  
MLS_range_set := MLS_range | {MLS_range MLS_range ...}  
MLS_range:= [:sensitivity[:category,...] [-sensitivity[:category,...]]]
```

role_set に user 名が遷移できる role を記述する。MLS_range_set にユーザに付与する MLS の範囲を記述する。MLS の範囲は複数記述できる。MLS の範囲は、左側に最下限の機密種別とオプションでカテゴリを、右側に最上限を記述する。最上限が記述されていない場合は最下限と同じになる。MLS の範囲が記述されていなければデフォルトで unclassified が付与される。

user 文で定義されていないユーザは、/etc/passwd ファイルにユーザのエントリがあったとしても、user 文で定義されない限りホストにログインすることができない。また/etc/security/default_context ファイルにログインするユー

ザのデフォルトの role、type を記述する必要がある。MLS が有効の場合は MLS の範囲も記述する。

新たにユーザに付加する type を宣言したときは login プログラムに定義した type に遷移できるように、domain/system/login.te ファイルに記述を追加する必要がある。

```
domain_trans(local_login_t, shell_exec_t, 新規の type)
```

そして、login プログラムの role である system_r が新しく宣言した type への遷移を許可するため、rbac ファイルの system_r のエントリに新しく宣言した type を追加する。

```
role system_r types {  
    新しく宣言した type  
}
```

また新しい role を定義して、その role でログインするユーザがいるならば、login プログラムの role である system_r が新しく宣言した role への遷移を許可するため、rbac ファイルに role 遷移許可ルールを記述する。

```
allow system_r 新しく宣言した role;
```

そして、/etc/security/default_type ファイルに新しく宣言した role のデフォルトの type を記述する。

2.1.6.8. Constraint の設定

constraint の設定では、TE と RBAC の設定で定義されたパーミッションに対して付加的な制約を定義する。この制約は TE や RBAC では設定できない細かい設定を行なうことができる。デフォルトでは constraints ファイルで定義されている。制約の定義は constrain 文を使う。以下に書式を示す。

```

constrain クラス名 パーミッション expression;
expression : ( expression )
            | not expression
            | expression and expression
            | expression or expression
            | u1 op u2
            | r1 role_op r2
            | t1 op t2
            | u1 op names
            | u2 op names
            | r1 op names
            | r2 op names
            | t1 op names
            | t2 op names

u1:source SID のユーザ ID  u2:target SID のユーザ ID
t1:source SID の type      t2:target SID の type
r1:source SID の role      r2: target SID の role
論理演算子 op : == | !=
role 演算子 role_op : == | != | eq | dom | domby | incomp
names : name | { name_list }
name_list : name | name_list name
name:ユーザ名、type 名、属性名、role 名

```

expression はブール表現で条件を表し、expression が真のときにクラス名に記述してあるクラスのパーミッションが許可される。expression では、ソース SID とターゲット SID のユーザ ID、role、type の比較式を記述する。または TE と RBAC で定義された特定のユーザ、type、role との比較式を記述する。使用できる演算子は、「同じである」を表す「==」と、「異なる」を表す「!=」である。

role_op は role に関する演算子であり、それぞれ以下に示す。ここで A、B は role である。

- A == B または A eq B A と B が同じ role である。
- A != B A と B は異なる role である。
- A dom B A は B に遷移できる。
- A domby B B が A に遷移できる。
- A incomp B A から B、B から A に遷移できない。

constraints ファイルで定義してある制約を示す。

```
constrain process transition
    ( u1 == u2 or t1 == privuser );
constrain dir_file_class_set { create relabelto relabelfrom }
    ( u1 == u2 or t1 == privowner );
```

1 番目の例ではセキュリティ ID の遷移に関する制約である。ソース SID のタイプに属性 privuser が付与されていなければ、ユーザ ID の変更を伴うセキュリティ ID の遷移は許されない。2 番目の例では、ディレクトリやファイルなどの再ラベル付けに関する制約である。ソース SID(サブジェクトの SID)のタイプに属性 privowner が付与されていなければ、他人のオブジェクト(サブジェクトと異なるユーザ ID のセキュリティ ID を持つオブジェクト)に対して再ラベル付けを行なうことは出来ない。

2.1.6.9. セキュリティコンテキストの設定

ここではセキュリティコンテキストの設定について説明する。

(1) 初期 SID コンテキスト

初期 SID コンテキストの設定では、システムの初期設定のために必要となるセキュリティコンテキストの SID をあらかじめ定義する。初期 SID コ

ンテキストは `initial_sid_contexts` ファイルに記述されている。設定書式は以下のとおりである。

```
sid SID 名 ユーザ ID:role:type[:MLS 範囲]
```

MLS 範囲は MLS が有効になっている場合にのみ記述する。

(2) ファイルシステムコンテキスト

セキュリティコンテキストをラベル付けしていないファイルシステムをマウントするときに、マウントする各ファイルにラベル付けするセキュリティコンテキストを定義する。デフォルトでは `fs_context` ファイルで定義されている。書式は以下のとおりである。

```
デバイスのメジャー番号   デバイスのマイナ番号  
                           ファイルシステムのセキュリティコンテキスト  
                           ファイルのセキュリティコンテキスト
```

記述のないデバイスの場合は初期 SID の `fs` と `file` の関連付けされたセキュリティコンテキスト (それぞれ `system_u:object_r:fs_t` と `system_u:object_r:file_t`) が使われる。また、`fs` と `file` はルートファイルシステムがラベル付けされていないときにも使われる。これはルートファイルシステムがマウントされる時点ではまだセキュリティサーバが初期化されていないからである。

(3) ネットワーク

ネットワークに関するセキュリティコンテキストの設定ではポート番号、ネットワークインタフェース、ノードそして NFS について記述する。現在

の設定ファイルでは AF_INET アドレスファミリのポートとアドレスのみサ
ポートしている。

(A) ポート番号

各ポートに付与するセキュリティコンテキスト設定は以下の書式であ
る。

```
(TCP | UDP) ポート番号[-ポート番号] セキュリティコンテキスト
```

TCP プロトコルまたはUDP プロトコルを指定できる。ポート番号を指定
する際にハイフンを用いることで、例えば 1024-2000 といったようにし
て、連続したポート番号に同じセキュリティコンテキストを付与できる。
ここで定義されていないポート番号にはデフォルトで定義されている初
期 SID の port(system_u:object_r:port_t)が付与される。以下に記述例
を示す。

```
tcp 21 system_u:object_r:ftp_port_t
```

例では、TCP プロトコルの 21 番ポートに付与するセキュリティコンテ
キストを定義している。

(B) ネットワークインタフェース

ネットワークインタフェースに付与するセキュリティコンテキスト設
定は以下の書式である。

例ではループバックアドレスで指定されるノード(自身のホスト)にセキュリティコンテキストを定義している。

(D) NFS

各 NFS サーバに付与するセキュリティコンテキスト設定は以下の書式である。

```
アドレス マスク ファイルシステムセキュリティコンテキスト
                                ファイルセキュリティコンテキスト
```

アドレスとマスクで指定した NFS サーバからマウントするファイルシステムにファイルシステムセキュリティコンテキストを、ファイルにファイルセキュリティコンテキストを付与する。NFS サーバのアドレスにマッチするエントリがなければ初期 SID の nfs (system_u:object_r:nfs_t)が付与される。以下に記述例を示す。

```
10.33.1.2 255.255.255.255
            system_u:object_r:nfs_t system_u:object_r:nfs_t
```

例ではアドレス 10.33.1.2 の NFS サーバが提供するファイルをマウントする際に付与するセキュリティコンテキストを定義している。

2.1.6.10. ファイルのラベル付け

ファイルにラベル付けする方法には setfiles プログラムの使用と chcon コマンドの2つの方法がある。ここではこれらの方法について説明する。

(1) setfiles

ホストのすべてのファイルにセキュリティコンテキストをラベル付けす

るには `setfiles` プログラムを使用する。このコマンドは `selinux/setfiles` ディレクトリにある。またこのディレクトリには、インストール時に使用した `file_contexts` ファイルがある。このファイルには、システム上のすべてのファイルにラベル付けするセキュリティコンテキストが記述されている。`setfiles` プログラムは以下の形式で使用する。

```
setfiles [-dnv] filename [path]
```

オプションの意味は以下のとおりである。

表 2.1-12 `setfiles` のオプション

オプション	説明
d	各ファイルにマッチした記述を表示する。
n	すでにラベル付けされているファイルに関しては再ラベル付けをしない。
v	ファイルのラベルを変更したら表示する。

`filename` にファイルパスとセキュリティコンテキストの対応が記述されている設定ファイルを、オプションの `path` でラベル付けを開始するパスを記述する。設定ファイルの書式は以下のとおりである。

```
filepath [-type] ( context | <<none>> )
```

`filepath` にファイルパスを記述する。`filepath` は正規表現を使って記述することができる。正規表現を `filepath` の中に記述すると、マッチするすべてのファイルとディレクトリに指定したセキュリティコンテキストが付与される。`context` でラベル付けするセキュリティコンテキストを指定する。セキュリティコンテキストには TE の設定で宣言されているものを指定しな

けらばならない。<<none>>はマッチするファイルに再ラベル付けしないことを示す。

オプション type は、特定のファイルタイプにだけラベル付けしたいときに使用する。使用できるのは ls コマンドのモードフィールドに表示されるファイルタイプと同じである。例えば、-d ならディレクトリのみ、-- なら通常のファイルのみにラベル付けをする。以下にすべてのオプション type を示す。

表 2.1-13 ファイルモード一覧

コード	種類
-	通常のファイル
b	ブロック特殊ファイル
c	文字特殊ファイル
d	ディレクトリ
l	シンボリックリンク
p	ネームドパイプ
s	ソケット

setfiles プログラムはファイルのラベル付けの対応が記述されたファイルの先頭のエン트리から処理していくので、1つのファイルに対して複数のエントリがあった場合は最後にマッチしたエントリのセキュリティコンテキストが付与される。また1つのファイルに複数のハードリンクがあり、各ハードリンクに異なるセキュリティコンテキストをラベル付けする記述になっていると、セキュリティコンテキストが<<none>>でない最後にマッチしたセキュリティコンテキストがファイルにラベル付けされる。以下に記述例を示す

```
/var(|.*) system_u:object_r:var_t  
/dev/fd[^/]* system_u:object_r:removable_device_t  
/usr/sbin/in\..*d system_u:object_r:inetd_child_exec_t
```

1 番目のエントリは、/var ディレクトリと/var ディレクトリ以下にあるすべてのファイルとサブディレクトリにマッチする。2 番目のエントリは /dev ディレクトリにあるファイルで、ファイル名の先頭が fd であるすべてのファイルにマッチする。3 番目のエントリは、in.ftpd など “in.” で始まり “d” で終わる任意の文字列にマッチする。

例として、/home/somebody ディレクトリ以下のすべてのファイルに対してセキュリティコンテキストの再ラベル付けの方法を示す。まず somebody_home_file_contexts ファイルを作成して以下のエントリを記述する。

```
/home/somebody(|/.*) somebody:user_r:user_t
```

次に setfiles プログラムによってラベル付けをする。

```
#!/setfiles somebody_home_file_contexts /home/somebody
```

上記のように新しくファイルを作成するのではなく file_contexts ファイルに追加してもよい。

(2) chcon

この chcon コマンドにより、ファイルのセキュリティコンテキストを変更することができる。以下に使い方を示す。

```

chcon [オプション] context file
chcon [オプション] --reference=RFILE file
chcon [オプション] --sid=SID file
    
```

context で変更後のセキュリティコンテキストを指定して file で変更対象となるファイル名を記述する。オプションを以下に示す。

表 2.1-14 chcon のオプション一覧

オプション	説明
c, --changes	再ラベル付けしたときだけ変更内容を表示する。
h, --no-dereference	シンボリックリンクのリンク先のファイルではなく、シンボリックリンクファイルのセキュリティコンテキストを変更する (lchown システムコールが利用できるシステムだけで利用可能である)。
f, --silent, --quiet	ほとんどのエラーメッセージを抑制する。
reference=RFILE	引数で指定する代わりに RFILE で指定されたファイルのセキュリティコンテキスト SID を使用する。
sid=SID	セキュリティコンテキストを SID で指定する。
R, --recursive	ファイルとディレクトリのラベル付けを再帰的にする。
v, --verbose	すべてのファイルについて、処理内容を出力する。
help	使用方法を表示する。
version	バージョン情報を表示する。

2.1.6.11. ポリシーの反映

ポリシー設定の変更や新しくルールを追加した場合は、ポリシー設定をコンパイルして「/」にコンパイルしたバイナリファイルのインストール、そしてセキュリティサーバが新しい設定のバイナリファイルをロードする必要がある。

(1) コンパイル

ポリシー設定のコンパイルは checkpolicy コマンドを使用する。以下に

書式を示す。

```
checkpolicy [-b] [-d] [-o 出力ファイル] [入力ファイル]
```

以下にオプションを説明する。

表 2.1-15 checkpolicy のオプション

オプション	説明
b	すでにコンパイルしたバイナリファイルを読み込む。
d	読み込んだポリシー設定を使って、セキュリティサーバの関数でユーザが対話的にテストできるようにする。
o	出力ファイルを指定する。指定しなければ policy となる。

入力ファイルを指定しなければ入力ファイルが policy.conf になる。入力ファイルは、TE や RBAC の設定ファイルを 1 つにまとめてマクロ展開したものでなければならない。実際にポリシー設定のコンパイルをするときはインストール時に利用した policy ディレクトリにある Makefile を使う。「make policy」と入力すれば、各種設定を 1 つにまとめてマクロ展開し、checkpolicy でコンパイルする。domain ディレクトリ下と types ディレクトリ下以外の場所に新しく設定ファイルを作成した場合は、Makefile に以下のように追加すればよい。

```
POLICYFILES += 作成したファイル名
```

domain ディレクトリと types ディレクトリ以下にあるファイルはすべて結合することになっているので Makefile に追加する必要はない。

設定ファイルのコンパイルはすべての設定ファイルを一つのファイルに結合したファイルに対して行なわれる。エラーが設定ファイル中にある場

合はそのファイルの行番号とエラーメッセージが出力される。そのため、設定言語の間違いが各種設定ファイルのどこで起こっているのかを突き止めるのが難しくもある。

(2) インストール

インストールではコンパイルした設定ファイルをセキュリティサーバが読み込むことができるようにするため、root ディレクトリにファイル名 `ss_policy` としてコピーする。インストールでも(1)と同様に `policy` ディレクトリにある `Makefile` を使う。

「`make install`」と入力すれば設定ファイルのコンパイルとインストールが行われる。インストールだけでは新しい設定ファイルをセキュリティサーバが読み込まないので再起動後に有効となる。

(3) ロード

ロードではセキュリティサーバに新しい設定ファイルをロードさせる。ロードさせるには `load_policy` コマンドを使用する。以下に書式を示す。

```
load_policy file
```

`file` にセキュリティサーバにロードされるバイナリ形式の設定ファイルのパスを記述する。ロードも今までと同様に `Makefile` を使って実行できる。

「`make load`」と入力すればコンパイル、インストールそしてロードが実行される。

2.1.7. SELinux の設定例

ここでは実際の設定例として、いくつかのサーバについての設定方法を説明する。また、ここではカレントディレクトリを `/usr/src/selinux/policy` ディレ

クトリとしてファイルの位置を記述している。

2.1.7.1. 設定の概要

SELinux で提供されている設定ファイルには、基本的なデーモンやサービスについての設定ファイルが domain/program と domain/system 以下に用意してある。例えば、Web サーバ Apache やメールサーバ Sendmail などがある。しかし、SELinux の設定ファイルに用意されていないデーモンなどのプログラムを動作させるには設定ファイルを新たに追加しなければならない。ここでは新たに設定ファイルを作成する際に必要となる知識について説明する。また、新しく設定ファイルを作成するときは、SELinux で用意されている設定ファイルの中で類似サービスの設定ファイルを参考にすると良い。

(1) 遷移の設定

(A) domain の遷移設定

新しく設定ファイルを作成する場合、作成対象となるプログラムは、同プログラムの起動を行なう親プロセスの domain とは別の domain にすることで権限分割をしたほうがよい。特にシステムプロセスの init や xinetd または inetd で起動されるプログラムは必ず別の domain で実行して権限分割をしなければならない。init や xinetd は複数のプログラムを起動させるのでさまざまな権限が付与されている。よって、プログラムが起動したときに domain の遷移を設定しなければ親プロセスと同じ domain となり、起動したプログラムに必要な権限を与えることになる。もし、このプログラムにセキュリティホールがあり攻撃者に乗っ取られた場合、攻撃者が init または xinetd と同じ権限を持つことになり大変危険である。domain の遷移設定による権限分割は重要となる。また、

domainの遷移設定をしないと親プロセスのdomainにも必要のない権限を与えることにもなる。

domainの遷移を設定する場合はinitプロセスによって起動、inetdまたはxinetdからの起動とユーザによる実行によって、各々、以下のファイルに遷移設定をする。

- init

domain/system ディレクトリにあるinitrc.te ファイル。また、init.te ファイルにも遷移の設定を記述する必要がある。

- xinetd または inetd

domain/system ディレクトリにあるinetd.te ファイル。

- ユーザによる実行

管理ユーザ sysadm_t ならば domain/system ディレクトリにある sysadm_t.te ファイル。一般ユーザ user_t ならば user.te ファイルである。それ以外ならばユーザのdomainが設定されているファイルに追加する。

- 上記以外

プログラムを実行するdomainの設定が記述されているファイルに追加する。

また、RBACの設定でもdomainの遷移を許可しておく必要がある。rbacファイルにて、プログラムを実行するプロセスに付与されるroleが宣言されている箇所に新しいdomainを追加する。

(B) オブジェクト type の遷移設定

domainが作成するファイルなどのオブジェクトについても遷移設定をしたほうがよい。/var ディレクトリ以下に作成されるファイルは、オブ

ジェクト type の遷移を設定しなければ type が var_t となる。この type は属性 domain を持つプロセスなら読み込み権限が与えられている。なるべくならば、新しく type を宣言してアクセス制限をしたほうがよい。type の遷移設定は domain の設定が記述してあるファイルに type_file_auto_trans マクロを使って記述すればよい。

(2) SELinux のログについて

新しい設定ファイルを作成する前に、SELinux のカーネルを Development mode で起動させて、設定ファイルの作成対象となるプログラムを実行させる。アクセス許可を全く与えてない状態でプログラムを実行させることで、そのプログラムに必要となるアクセス権限がログファイルを見ることわかる。そこで SELinux が出力するログの見方について説明する。

(A) ログの内容

デフォルトではポリシー設定によって許可されていないアクセスが発生したときに、カーネルメッセージとして /var/log/messages ファイルにログが出力される。TE の設定で auditallow 文を使ってアクセスが許可されたときにもログを出力できる。以下にアクセスが拒否されたときと許可されたときに出力されるログの例を示す。

アクセス拒否

```
avc:denied {write} for pid=5485 exe=/bin/bash  
path=/root/.bash_history dev=03:01 ino=355699  
scontext=takeuchi:user_r:user_t:unclassified
```

```
tcontext=system_u:object_r:sysadm_home_t:unclassified  
tclass=file
```

アクセス許可

```
avc: granted {setuid}for pid=4964  
exe=/usr/sbin/xinetd  
capability=7
```

```
scontext=system_u:system_r:inetd_t:unclassified  
tcontext=system_u:system_r:inetd_t:unclassified  
tclass=capability
```

まず、アクセスを拒否したときのログについて説明する。「avc」は access vector cache の略であり、avc: はアクセスベクタキャッシュが出力したログを意味する。「denied {write}」は拒否されたパーミッションを表す。この場合は write パーミッションである。もし、denied の代わりに granted が出力されたならば許可されたパーミッションを表す。「for pid=5485 exe=/bin/bash」がサブジェクトの情報である。例ではプロセス番号 5485 で /bin/bash を実行しているプロセスとなる。「path=/root/.bash_history dev=03:01 ino=355699」がオブジェクトの情報である。例では、デバイスのメジャー番号 3、マイナ番号 1、inode355699 である /root/.bash_history ファイルを示している。scontext がソース type に付与されているセキュリティコンテキストを表す。tcontext がターゲット type に付与されているセキュリティコンテキストを表す。

このログは user_t が sysadm_home_t への write 権限が許可されていないことを示している。例えば、このログで拒否されている権限を与えるためには以下のように定義すればよい。

```
allow user_t sysadm_home_t file:write
```

アクセスが許可されたときのログについて説明する。「granted { setuid }」が許可したパーミッションを表す。「setuid」が許可されたパーミッションとなる。「capability=7」は、各 Linux Capability に割り振られている番号である。/usr/include/linux/capability.h ファイルに各 Capability に割り振られている番号を見ることができる。整数 7 は、CAP_SETUID であるので setuid システムコールに関する Capability である。scontext と tcontext は拒否のログと同様である。例では sccontext と tcontext が同じとなっている。これは capability setuid がプロセス自身に与えられる権限だからである。このログでは、domain が inetd_t であるプロセスがシステムコール setuid を使用したことを示している。

(B) ログから allow 文の作成

SELinux ではアクセスが拒否されたログから allow 文を作成ツールが提供されている。このツールを使用することで新しい domain を定義する際にどのようなアクセス権限を与えたら良いか分かる。このツールは、/var/log/messages ファイルに出力されたログから allow 文を作成している。

allow 文を作成するツールは script ディレクトリにある newrules.pl ファイルである。このスクリプトは Perl で書かれているので使用する際には、マシンに Perl がインストールされている必要がある。使用方法は引数なしでスクリプトを実行すればよい。

(3) 設定ファイルの作成

新規に設定ファイルを作成する際には、ユーザが実行するプログラムの設定ファイルは domain/program ディレクトリに、init や xinetd などのシステムプロセスによって実行されるプログラムの設定ファイルは domain/sysadm ディレクトリに作成するようにする。

(4) 属性とマクロの利用

新しく定義した domain にアクセス許可を与えるときは、属性やマクロを使って定義するようにしたほうがよい。また、新規にマクロと属性を定義することも考えるとよい。属性とマクロを利用することで記述が簡単になり、また、設定ファイルが理解しやすいものになる。

(5) ネットワークの設定

domain にネットワーク利用を許可させるために can_network マクロを利用する。このマクロを利用することでネットワークインタフェースを使用する権限や、メッセージの送受信に関する権限が許可される。

また、特定のポートだけの利用を許可する方法は以下のとおりである。

(A) type 宣言

types/network.te ファイルに、利用するポートの type を port_type 属性を付加して宣言する。

```
type type 名, port_type;
```

(B) type への関連付け

(A)で宣言した type を、使用するプロトコルとポート番号に関連付け

するために、net_contexts ファイルでポート番号のセキュリティコンテキストを定義する。

```
tcp ポート番号 system_u:object_r:type 名
```

プロトコルを UDP にする場合は、“tcp”を“udp”にする。

(c) アクセスベクタルールの設定

TE の設定で宣言した type へのアクセス権限を許可する。アクセス先が tcp のポートなので、クラス tcp_socket の name_bind パーミッションを許可する。

```
allow ドメイン名 ポートのタイプ名:tcp_socket name_bind;
```

また、1023 以下の特権ポートを使用するにはクラス capability の net_bind_service パーミッションが必要となる。

(6) SELinux のコマンド

設定の際に利用する SELinux のコマンドを説明する。

(A) avc_toggle

SELinux のカーネルオプション「NSA SELinux Development Module」を有効にしたカーネルで起動しているときに、「enforcing mode」と「permissive mode」を切り替えるコマンドである。引数なしでコマンドを実行することで、「permissive mode」で動作していれば「enforcing mode」となり、「enforcing mode」で動作していれば「permissive mode」となる。

(B) newrole

指定した role でシェルを新規に実行するコマンドである。例えば管理者 role の sysadm_r でログインしたときに、一般ユーザ role の user_t になって設定を確認したいときなどに使用する。使用方法は以下のとおりである。

```
newrole [-r|--role] role 名 [ [シェル引数]...]
newrole [-t|--type] type 名 [ [シェル引数]...]
newrole [-r|--role] role 名 [-t|--type] TYPE
                                     [ [シェル引数]...]
```

上記のように指定した role または type でシェルが起動される。“シェル引数”とは、シェルを起動するときに渡す引数である。

(C) runas

指定したセキュリティコンテキストまたはSIDでコマンドを実行する。使用方法は以下のとおりである。

```
runas [-t type 名] [-l MLS のレベル] [-u ユーザ ID]
                                           [-r role 名] コマンド [引数]
runas コンテキスト コマンド [引数]
runas -s SID コマンド [引数]
```

(D) run_init

rc スクリプトを、/etc/security/initrc_context ファイルで指定されたセキュリティコンテキストで実行するコマンドである。rc スクリプトで起動されるデーモンなどの設定ファイルを変更したときに、このコマンドを使用して設定を確認する。以下に使用方法を示す。

```
run_init スクリプト名 [[引数]....]
```

“スクリプト名”で実行するスクリプトを指定し、必要に応じて実行するスクリプトの“引数”を指定する。

2.1.7.2. Web サーバ

Web サーバ Apache の設定例を取り上げる。使用した Apache はバージョン 1.3.20 である。ここで説明する設定ファイルはデフォルトで用意されている domain/system/apache.te ファイルとは異なっており、説明のために作成した設定ファイルである。設定ファイルは、domain/system/apache_sample.te ファイルとして作成した。

Apache の httpd デーモンの domain を httpd_sample_t として起動させ、httpd の実行ファイルの type を httpd_exec_sample_t とする。本設定では、Web サーバで静的コンテンツのみを扱う場合の設定を行なう。また、管理専用ユーザ「webadmin」を追加して Web コンテンツなどの管理をさせる。また、Web 管理者が扱うコンテンツと一般ユーザが扱うコンテンツを以下のように分ける。

(A) Web 管理者

- Web コンテンツ領域である /var/www ディレクトリ以下の html ファイルの type を httpd_sys_content_sample_t とする。

(B) 一般ユーザ

- 一般ユーザの public_html ディレクトリに作成される html ファイルの type を httpd_user_content_sample_t とする。

また、Web サーバからは html ファイルに対して read のみを許可し、Web 管理者のみがすべての Web コンテンツへの read、write を許可する。これらのポリ

シーに基づいて設定ファイルを作成していく。

(1) ネットワークの設定

まず、Web サーバが使用するポートに付与する type を宣言する。
types/network.te ファイルに以下を追加する。

```
type httpd_port_t port_type;
```

次に、httpd_port_t と Web サーバが使用する TCP80 番ポートを関連付け
するために net_contexts ファイルに以下を追加する。

```
tcp 80 system_u:object_r:httpd_port_t
```

以上によって、TCP80 番ポートを使用するには httpd_port_t へのアクセ
ス許可が必要となる。

(2) TE の設定

TE の設定ではまず type 宣言をする。最初に Apache の httpd デーモンに
付与する domain と httpd の実行ファイルに付与する type を宣言する。
httpd_sample_t は、Apache のデーモンに付与するので、domain 属性を付与
する。httpd_exec_sample_t は実行ファイルに付与するので属性 file_type
と属性 exec_type を付与する。また、管理者によってのみフルアクセスで
きるように属性 sysadmfile も付与する。

次に httpd_sample_t がアクセスするファイルにラベル付けする type を
宣言する。pid ファイル、Apache の設定ファイルをそれぞれ定義する。ま
た、pid ファイルの type には pid ファイルを示す属性 pidfile を付与する。
pid ファイルが作成されたとき、type 遷移をさせるために

`file_type_auto_trans` マクロを使う。

`httpd` デーモンでは、いくつかの `capability` が必要なので必要な `capability` を `httpd` 自身に付与する。

`httpd` は、`init` から起動されるので `init` からファイルディスクリプタを引き継ぐ必要がある。

ネットワークを使用できるようにするために `can_network` マクロによって必要な権限を与える。

また、`httpd` が使用するポートに TCP ソケットをバインドしなければならないので HTTP ポートへのバインドを許可する。

ログファイルへの書き込み許可と、`pid` ファイルへの書き込み許可を与える。

Web コンテンツへのアクセス許可を設定する。

`httpd` を管理者によって起動させたい場合は管理者 `domain` の `sysadm_t` からの遷移を設定する。

以下に実際の設定ファイルを示す。

```
#domain, type の宣言
type httpd_sample_t, domain, privlog;
type httpd_exec_sample_t, file_type, sysadmfile, exec_type;
type httpd_config_sample_t, file_type, sysadmfile;
#WEB コンテンツの type
type httpd_sys_content_sample_t, file_type;
type httpd_user_content_sample_t, file_type;
#pid ファイル、log 用の type の宣言、遷移設定
type httpd_var_log_sample_t, file_type, sysadmfile;
type httpd_var_run_sample_t, file_type, sysadmfile, pidfile;
#/var/log/に書き出すときは、httpd_var_log_sample_t に変換
file_type_auto_trans(httpd_sample_t, var_log_t,
                      httpd_var_log_sample_t)
file_type_auto_trans(httpd_sample_t, var_run_t,
                      httpd_var_run_sample_t)

#ケイパビリティの設定
allow httpd_sample_t httpd_sample_t:capability
                      {chown net_bind_service setgid setuid kill};
#/dev/console 出力に使う fd を init から継承
allow httpd_sample_t init_t:fd inherit_fd_perms;
#ネットワーク使用
can_network(httpd_sample_t)
# http ポートの使用
allow httpd_sample_t http_port_t:tcp_socket name_bind;
#httpd は/var/run/に pid を書く
allow httpd_sample_t var_run_t:file rw_file_perms;
allow httpd_sample_t var_run_t:file create_file_perms;
allow httpd_sample_t var_run_t:file link_file_perms;
allow httpd_sample_t var_run_t:dir rw_dir_perms;
#/var/log/ にログを書き出すための設定
allow httpd_sample_t var_log_t:file rw_file_perms;
allow httpd_sample_t var_log_t:file create_file_perms;
allow httpd_sample_t var_log_t:file link_file_perms;
allow httpd_sample_t var_log_t:dir rw_dir_perms;
#/var/www 以下を見ることがするための設定
allow httpd_sample_t httpd_sys_content_sample_t:
                      file r_file_perms;
allow httpd_sample_t httpd_sys_content_sample_t:
                      dir r_dir_perms;
allow httpd_sample_t httpd_user_content_sample_t:
                      file r_file_perms;
allow httpd_sample_t httpd_user_content_sample_t:
                      dir r_dir_perms;

# sysadm_t で実行させるとき
domain_auto_trans(sysadm_t, httpd_exec_sample_t,
                  httpd_sample_t)
```

httpd は rc スクリプトから起動されるので httpd を実行したときに、domain 遷移させるようにする。rc スクリプトを実行する init プロセスの domain は initrc_t なので、httpd_exec_sample_t を実行したときに httpd_sample_t に遷移させるように宣言する。initrc_t の定義は、domain/system/initrc.te なので以下を追加する。

```
domain_auto_trans(initrc_t,httpd_exec_sample_t,  
                  httpd_sample_t)
```

(3) RBAC の設定

先に initrc_t から httpd_sample_t への遷移を許可したが、initrc_t の role でも httpd_sample_t への遷移を許可しなければならない。initrc_t の role は system_r である。system_r が httpd_sample_t に遷移できるようにするため、system_r に httpd_sample_t を加える。rbac ファイルに以下を追加する。

```
role system_r types {  
    .....  
    httpd_sample_t  
    .....  
}  
# 管理者が起動させるとき。  
role sysadm_r types {  
    .....  
    httpd_sample_t  
    .....  
}
```

(4) Web 管理者の設定および一般ユーザの設定

Web 管理者の domain 設定を記述する domain/user/webadmin.te ファイル

を作成する。

まず管理者 webadmin の domain を宣言し、一般ユーザとして必要な権限を与えるために user_domain (domain 名) マクロを使用する。

次に Apache の設定ファイルとログファイルへの読み書き権限を与える。そして、すべての Web コンテンツへのアクセス権限を与える。

```
# domain、type の宣言。
type httpd_admin_sample_t, domain, userdomain;
一般ユーザの権限を与える。
user_domain(httpd_admin_sample)
#設定ファイルとログファイルのアクセス権。
allow httpd_admin_sample_t httpd_config_sample_t:
    file create_file_perms;
allow httpd_admin_sample_t httpd_config_sample_t:
    dir rw_dir_perms;
allow httpd_admin_sample_t httpd_config_sample_t:
    lnk_file link_file_perms;
allow httpd_admin_sample_t httpd_config_sample_t:
    security { sid_to_context };
allow httpd_admin_sample_t httpd_log_files_sample_t:
    file create_file_perms;
allow httpd_admin_sample_t httpd_log_files_sample_t:
    dir rw_dir_perms ;
allow httpd_admin_sample_t httpd_log_files_sample_t:
    security { sid_to_context };
allow httpd_admin_sample_t httpd_log_files_sample_t:
    lnk_file link_file_perms;

# Web コンテンツへのアクセス権を設定。
allow httpd_admin_sample_t httpd_sys_content_sample_t:
    file create_file_perms;
allow httpd_admin_sample_t httpd_user_content_sample_t:
    file create_file_perms;
allow httpd_admin_sample_t httpd_sys_content_sample_t:
    dir rw_dir_perms;
allow httpd_admin_sample_t httpd_user_content_sample_t:
    dir rw_dir_perms;
allow httpd_admin_sample_t httpd_sys_content_sample_t:
    lnk_file {create};
allow httpd_admin_sample_t httpd_user_content_sample_t:
    lnk_file {create};
```

Web 管理者の role を httpd_admin_sample_r として rbac ファイルに宣言し、遷移可能な domain を記述する。ここで、Web 管理者など新しく role を

作成したときに、一般ユーザとして必要な domain を定義する
user_role_type マクロを作成し利用する。

```
role httpd_admin_sample_r types {
    user_role_type(httpd_admin_sample)
};

define(`user_role_type',`
    $1_t
    passwd_t
    utempter_t
    newrole_t
    $1_lpr_t
    $1_mail_t
    $1_xserver_t
    $1_gph_t
    $1_su_t
    $1_netscape_t
    $1_crontab_t
    $1_crond_t
    $1_ssh_t
`)
```

また、rbac ファイルに login プログラムの role である system_r が
webadmin ユーザの role と type に遷移できるようにする。

```
role system_r types {
    httpd_admin_sample_t
}

allow system_r httpd_admin_sample_r;
```

user ファイルで webadmin がログインできるように設定する。

```
user webadmin role{httpd_admin_sample_r};
```

webadmin がログインするときのデフォルトの role と domain を

/etc/security/default_context に記述する。

```
webadmin:httpd_admin_sample_r:httpd_admin_sample_t
```

また、httpd_admin_sample_r のデフォルトの type を /etc/security/default_type に記述する。

```
httpd_admin_sample_r:httpd_admin_sample_t
```

type が httpd_admin_sample_t のシェルは login プログラムから webadmin がログイン時に立ち上がるので、login プログラムの domain 設定が記述された login.te ファイルで遷移設定を行なう。domain/system/login.te ファイルに以下を追加する。

```
domain_trans(local_login_t, shell_exec_t,  
             httpd_admin_sample_t)
```

また、一般ユーザに httpd_user_content_sample_t へのアクセス権限を与えるために domain/user/user.te に以下を追加する。

```
# Web コンテンツへのアクセス権を設定。  
allow user_t httpd_user_content_sample_t:  
           file create_file_perms;  
allow user_t httpd_user_content_sample_t:  
           dir rw_dir_perms;  
allow user_t httpd_user_content_sample_t:  
           lnk_file {create};
```

(5) ファイルのラベル付け

TE の設定で宣言した type をファイルおよびディレクトリに対して再ラベ

ル付けする。

```
/var/www/html(|/.*)  
    system_u:object_r:httpd_sys_content_sample_t  
/var/www/icons(|/.*)  
    system_u:object_r:httpd_sys_content_sample_t  
/etc/httpd    system_u:object_r:httpd_config_sample_t  
/etc/httpd/conf(|/.*)  
    system_u:object_r:httpd_config_sample_t  
/etc/httpd/logs  
    system_u:object_r:httpd_log_files_sample_t  
/usr/sbin/httpd  
    system_u:object_r:httpd_exec_sample_t  
/var/log/httpd(|/.*)  
    system_u:object_r:httpd_log_files_sample_t  
/home/.*/*public_html(|/.*)  
    system_u:object_r:httpd_user_content_sample_t  
/home/webadmin(|/.*)  
    system_u:object_r:httpd_admin_sample_home_t
```

2.1.7.3. DNS サーバ

DNS サーバ BIND の設定を取り上げる。使用した BIND はバージョン 9.1.3 である。ここではデフォルトの設定ファイルで用意されている `policy/domain/system/named.te` ファイルを使って説明する。

BIND のポリシー設定では新たに BIND の `named` デーモンのプロセスに付与する `domain` を `named_t` として宣言し、`named` がアクセスするオブジェクトと設定ファイルの `type` も宣言する。また、`named` が使用するネットワークについても `type` を宣言する必要がある。

(1) ネットワークの設定

まず、DNS サーバが使用するポートに付与する `type` を宣言する。
`types/network.te` ファイルに以下を追加する。

```
type named_port_t, port_type;
```

次に、named_port_t と DNS サーバが使用する TCP と UDP の 53 番ポートを
関連付けするために net_contexts ファイルに以下を追加する。

```
tcp 53 system_u:object_r:named_port_t;  
udp 53 system_u:object_r:named_port_t;
```

これによって TCP と UDP の 53 番ポートを使用するには、named_port_t へ
のアクセス許可が必要となる

(2) TE の設定

TE の設定ではまず type の宣言をする。最初に DNS の named デーモンに付
与する domain と、named の実行ファイルに付与する type を宣言する。
named_t は DNS のデーモンに付与するので、domain 属性を付与する。また、
named はログを syslogd で出力するので、それを許可する privlog 属性も付
与する。named_exec_t は実行ファイルに付与するので、属性 file_type、
exec_type を付与する。また、管理者によってのみフルアクセスできるよう
に sysadmfile も付与する。

次に、named_t がアクセスするファイルにラベル付けするオブジェクト
type を宣言する。pid ファイルと DNS の設定ファイルをそれぞれ定義する。
また、pid ファイルの type には pid ファイルを示す属性 pidfile を付与す
る。pid ファイルが作成されたときに type 遷移をさせるために
file_type_auto_trans マクロを使う。

named はいくつかの capability が必要なので、必要な capability を named

自身に付与する。

named は、init から起動されるので init からファイルディスクリプタを引き継ぐ必要がある。

named がネットワークを使用できるようにするために can_network マクロによって必要な権限を与える。

また、named が使用するポートに TCP ソケットと UDP ソケットをバインドしなければならないので DNS ポートへのバインドを許可する。

ログファイルへの書き込み許可と DNS 設定ファイルへの読み込み許可を与える。

named を管理者によって起動させたい場合は管理者 domain の sysadm_t からの遷移を設定する。

以下に実際の設定ファイルを示す。

```
#named デーモンの domain、type の宣言
type named_t, domain, privlog;
type named_exec_t, file_type,sysadmfile, exec_type;
#設定ファイル、pid ファイルの type 宣言
type named_conf_t, file_type,sysadmfile;
type named_var_run_t, file_type, sysadmfile,pidfile;
file_type_auto_trans(named_t, var_run_t, named_var_run_t)
# named に必要な capabilities の付与
allow named_t named_t:capability { setuid setgid
net_bind_service };
# init プロセスから fd を引き継ぎ、使用する。
allow named_t init_t:fd inherit_fd_perms;
#ネットワーク利用許可
can_network(named_t)
# named が DNS ポートを利用できる許可
allow named_t named_port_t:{udp_socket tcp_socket}
name_bind;

#ログファイルへの書き込み許可
allow named_t var_log_t:file ra_file_perms;
#DNS の設定ファイルの読み込み許可
allow named_t named_conf_t:file r_file_perms;
allow named_t named_conf_t:dir search;
# sysadm_t で実行させるとき。
domain_auto_trans(sysadm_t, named_exec_t,named_t)
```

named は rc スクリプトから起動されるので named を実行したときに、domain 遷移させるようにする。rc スクリプトに付与する domain は initrc_t なので、named_exec_t を実行したときに named_t に遷移させるように宣言する。initrc_t の定義は、domain/system/initrc.te なので以下を追加する。

```
domain_auto_trans(initrc_t, named_exec_t, named_t)
```

(3) RBAC の設定

先に initrc_t から named_t への遷移を許可したが、initrc_t の role でも named_t への遷移を許可しなければならない。initrc_t の role は system_r である。system_r が named_t に遷移できるようにするため system_r に named_t を加える。rbac ファイルに以下を追加する。また、管理者によって起動させたいときは管理者 domain sysadm_t にも追加する。

```
role system_r types {
    named_t
}
# 管理者が起動させるとき。
role sysadm_r types {
    named_t
}
```

(4) ファイルのラベル付け

TE の設定で定義した type を、ファイルおよびディレクトリに対してラベ

ル付けする。

```
/var/named(|/.*) system_u:object_r:named_conf_t  
/etc/named.conf system_u:object_r:named_conf_t  
/usr/sbin/named system_u:object_r:named_exec_t
```

2.1.7.4. メールサーバ

メールサーバ sendmail の設定を取り上げる。使用した sendmail はバージョン 8.11.6 である。ここでは、デフォルトの設定ファイルで用意されている policy/domain/system/sendmail.te ファイルを使って説明する。

sendmail のポリシー設定では新たに sendmail のプロセスに付与する domain sendmail_t を宣言し、sendmail がアクセスするオブジェクトとメール送受信用のメールプールやメッセージキューなどの type も設定する。また、sendmail が使用するネットワークについても type を宣言する必要がある。

(1) ネットワーク設定

まず、sendmail が使用するポートの type を宣言する。ファイル types/network.te に以下を追加する。

```
type smtp_port_t, port_type;
```

次に、smtp_port_t と tcp の 25 番ポートを関連付けするために、ファイル net_contexts に以下を追加する。

```
tcp 25 system_u:object_r:smtp_port_t
```

これによって tcp の 25 番ポートを使用するには、smtp_port_t へのアク

セス許可が必要となる。

(2) TE の設定

TE の設定ではまず type の宣言を行なう。最初に sendmail プロセスに付与する domain と sendmail の実行ファイルに付与する type を宣言する。sendmail_t は、sendndmail の実行プロセスに付与するので domain 属性を付与する。また、sendmail は syslogd を使ってログを出力するので、それを許可する privlog 属性も付与する。sendmail_exec_t は sendmail 実行ファイルに付与するので属性 file_type と属性 exec_type を付与する。また、管理者によってのみフルアクセスできるように sysadmfile も付与する。

次に、sendmail_t がアクセスするファイルにラベル付けするオブジェクト type を宣言する。テンポラリファイル、pid ファイルそして log ファイルをそれぞれ宣言する。また、テンポラリファイルの type にはテンポラリファイルを示す属性 tmpfile を、pid ファイルの type には pid ファイルを示す属性 pidfile をそれぞれ付与する。それらのファイルが作成されたときに、type 遷移をさせるために file_type_auto_trans マクロを使う。

sendmail ではいくつかの capability が必要なので、必要な capability を sendmail_t に付与する。

sendmail は、init から起動されるので init からファイルディスクリプタを引き継ぐ必要がある。

sendmail_t がネットワークを使用できるようにするため can_network マクロによって必要な権限を与える。

また、sendmail が使用するポートに TCP ソケットをバインドしなければならないので SMTP ポートへのバインドを許可する。

sendmail は受信したメールと再送するためのメールを保存するため、

メールプールとメッセージキューへのアクセス権限を設定する。

sendmail を管理者によって起動させたい場合は、管理者 domain の sysadm_t からの遷移を設定する。

```
#sendmail の domain,type の宣言
type sendmail_t, domain, privlog;
type sendmail_exec_t, file_type, sysadmfile, exec_type;
#tmp,pid,log ファイルの宣言と作成したときに遷移するようにする。
type sendmail_tmp_t, file_type, sysadmfile, tmpfile;
file_type_auto_trans(sendmail_t, tmp_t, sendmail_tmp_t)
type sendmail_var_log_t, file_type, sysadmfile;
file_type_auto_trans(sendmail_t,var_log_t,
                      sendmail_var_log_t)
type sendmail_var_run_t, file_type, sysadmfile, pidfile;
file_type_auto_trans(sendmail_t, var_run_t,
                      sendmail_var_run_t)

#sendmaill に必要な capability を付与する。
allow sendmail_t sendmail_t:capability { setuid setgid
                                         net_bind_service sys_nice chown };

#init プロセスから fd を引き継ぎ、使用する。
allow sendmail_t init_t:fd inherit_fd_perms;
# ネットワーク利用許可
can_network(sendmail_t)
#SMTP ポートにバインドする許可
allow sendmail_t smtp_port_t:tcp_socket name_bind;
#/etc/aliases と/etc/mail への書き込み権限を与える。
allow sendmail_t etc_aliases_t:file rw_file_perms;
allow sendmail_t etc_mail_t:dir rw_dir_perms;
allow sendmail_t etc_mail_t:file create_file_perms;
# /var/spool/mail と/var/spool/mqueue への書き込み許可
allow sendmail_t mail_spool_t:dir rw_dir_perms;
allow sendmail_t mail_spool_t:file create_file_perms;
allow sendmail_t mqueue_spool_t:dir rw_dir_perms;
allow sendmail_t mqueue_spool_t:file create_file_perms;
# sysadm_t で実行させるとき。
domain_auto_trans(sysadm_t, sendmail_exec_t,sendmail_t)
```

sendmail は rc スクリプトから起動されるので、sendmail を実行したときに sendmail_t のドメインに遷移させるようにする。rc スクリプトに付与する domain は、initrc_t なので sendmail を実行したときに sendmail_t に

遷移させるように定義する。initrc_t の定義は domain/system/initrc.te なので以下を追加する。

```
domain_auto_trans(initrc_t,sendmail_exec_t,sendmail_t)
```

(3) RBAC の設定

先に initrc_t から sendmail_t への遷移を許可したが、initrc_t の role でも sendmail_t への遷移を許可しなければならない。initrc_t の role は system_r である。system_r に sendmail_t に遷移できるようにするために system_r に sendmail_t を加える。RBAC の設定ファイル rbac に以下を追加する。また、管理者によって起動させたいときは管理者 domain の sysadm_r にも追加する。

```
role system_r types {
    sendmail_t
}
# 管理者が起動させるとき。
role sysadm_r types{
    sendmail_t
}
```

(4) ファイルのラベル付け

TE の設定で宣言した type を、ファイルおよびディレクトリに対して再ラベル付けする。

```
/usr/sbin/sendmail      system_u:object_r:sendmail_exec_t
/var/log/sendmail.st    system_u:object_r:sendmail_var_log_t
/var/spool/mail(|/.*)   system_u:object_r:mail_spool_t
/var/spool/mqueue(|/.*) system_u:object_r:mqueue_spool_t
```

2.1.7.5. POP サーバ

POP サーバ qpopper の設定例を取り上げる。使用した qpopper はバージョン 4.0.3 である。ここで説明する設定ファイルは、デフォルトで用意されていないので新規に `policy/domain/system/pop.te` を作成した。

qpopper のポリシー設定では新たに qpopper のプロセスに付与する domain popper_t を宣言する。qpopper は、メールサーバが受信したメールを取り出すのでメール受信用のメールスプールへのアクセス権限が必要となる。また、qpopper が使用するネットワークについても type を宣言する必要がある。

(1) ネットワークの設定

まず popper が使用するポートの type を宣言する。ファイル `types/network.te` に以下を追加する。

```
type popd_port_t, port_type;
```

次に `popd_port_t` と tcp の 110 番ポートを関連付けするために、ファイル `net_contexts` に以下を追加する。

```
tcp 110 system_u:object_r:popd_port_t
```

これによって tcp の 110 番ポートを使用するには、`popd_port_t` へのアクセス許可が必要となる。

(2) TE の設定

TE の設定ではまず type の宣言を行なう。最初に qpopper プロセスに付与する domain と qpopper の実行ファイルに付与する type を宣言する。popper_t は、qpopper の実行プロセスに付与するので domain 属性を付与する。また qpopper は syslogd を使ってログを出力するのでそれを許可する privlog 属性と、認証する際に/etc/shadow にアクセスするので属性 auth も付与する。popper_exec_t は qpopper 実行ファイルに付与するので属性 file_type と属性 exec_type を付与する。また管理者によってのみフルアクセスできるように sysadmfile も付与する。

qpopper はいくつかの capability が必要なので、必要な capability を popper_t に付与する。

qpopper は xinetd から起動されるので xinetd からファイルディスクリプタを引き継ぐ必要がある。

引き継いだファイルディスクリプタのアクセス許可を与える。

qpopper はメールを取り出す必要があるので、メールプールへのアクセス権限を設定する。以下に実際の設定を示す。

```
#domain と type の宣言
type popper_t, domain, privlog, auth;
type popper_exec_t, file_type, sysadmfile, exec_type;
#ケイパビリティの設定
allow popper_t popper_t:capability { setuid
                                     setgid net_bind_service sys_nice chown };
# inetd からファイルディスクリプタを引き継ぐ。
# inetd のファイルディスクリプタは、ソケット用なので引き継ぐ必要がある。
allow popper_t inetd_t:fd inherit_fd_perms;
# inetd で開いたソケットを使う。
allow popper_t inetd_t:tcp_socket rw_stream_socket_perms;
# プロセスが終了したとき、sigchld を xinetd に送る。
allow popper_t inetd_t:process sigchld;
#/var/spool/mail 以下を読み書きできる。
```

```
allow popper_t mail_spool_t:file create_file_perms;  
allow popper_t mail_spool_t:dir create_dir_perms;  
allow popper_t mail_spool_t:file rw_file_perms;  
allow popper_t mail_spool_t:dir rw_dir_perms;
```

qpopper は xinetd から起動されるので、xinetd が qpopper を実行したときに popper_t の domain に遷移させるようにする。また、xinetd に popper が使用する 110 番ポートへのアクセスを許可する。xinetd に付与する domain は、inetd_t なので inetd_t の定義ファイル domain/system/inetd.te に以下を追加する。

```
domain_auto_trans(inetd_t, popper_exec_t, popper_t)  
allow inetd_t popd_port_t:tcp_socket:name_bind;
```

(3) RBAC の設定

先に、inetd_t から popper_t への遷移を許可したが、inetd_t の role でも popper_t への遷移を許可しなければならない。inetd_t の role は system_r である。system_r に popper_t に遷移できるようにするため、system_r に popper_t を加える。RBAC の設定ファイル rbac に以下を追加する。

```
role system_r types {  
    .....  
    popper_t  
    .....  
}
```

(4) ファイルのラベル付け

TE の設定で宣言した type を、ファイルおよびディレクトリに対してラベル付けする。

```
/usr/local/lib/popper      system_u:object_r:popper_exec_t
```

2.1.7.6. OpenSSH

OpenSSH によるリモートログインの設定を取り上げる。使用した OpenSSH はバージョン 2.9.p2 である。ここではデフォルトの設定ファイルで用意されている `policy/domain/system/sshd.te` ファイルを使って説明する。OpenSSH のポリシー設定では、新たに OpenSSH のプロセスに付与する domain の `sshd_t` を宣言し、OpenSSH がアクセスするオブジェクト、通信路の暗号化で使用する key に付加する type も宣言する。また、OpenSSH が使用するネットワークについても type を宣言する必要がある。

(1) ネットワーク設定

まず OpenSSH が使用するポートの type を宣言する。ファイル `types/network.te` に以下を追加する。

```
type sshd_port_t, port_type;
```

次に、`sshd_port_t` と `tcp` の 22 番ポートを関連付けするためにファイル `net_contexts` に以下を追加する。

```
tcp 22 system_u:object_r:sshd_port_t
```

これによって、tcp の 22 番ポートを使用するには `sshd_port_t` へのアクセス許可が必要となる。

(2) TE の設定

TE の設定ではまず `type` の宣言をする。最初に `sshd` プロセスに付与する `domain` と `sshd` の実行ファイルに付与する `type` を宣言する。`sshd_t` は、`sshd` の実行プロセスに付与するので `domain` 属性を付与する。また、`sshd` は `syslogd` を使ってログを出力するのでそれを許可する `privlog` 属性を付与する。そして、`sshd` はログインするユーザに応じて、`role`、ユーザ ID と端末のセキュリティコンテキストを変更しなければならないので、それぞれを許可する `privuser` 属性、`privrole` 属性と `privowner` 属性を付与する。`sshd` はログインプログラムを起動するので、`sshd` が起動する場合は別の `domain` である `sshd_login_t` とする。これにも `sshd_t` と同じ属性を付与する。また認証する際に `/etc/shadow` にアクセスするので `auth` 属性も付与する。`sshd_exec_t` は `sshd` 実行ファイルに付与するので、`file_type` 属性と `exec_type` 属性を付与する。また、管理者によってのみフルアクセスできるように `sysadmfile` も付与する。`sshd_key_t` は、通信路の暗号化に使用する `key` に付与する `type` である。

`sshd` ではいくつかの `capability` が必要なので、必要な `capability` を `sshd_t` に付与する。

`sshd` は、`init` から起動されるので `init` からファイルディスクリプタを引き継ぐ必要がある。

`sshd` がネットワークを使用できるようにするため、`can_network` マクロによって必要な権限を与える。

ユーザの端末を作成するために `ptys` の作成許可が必要となる。

sshd が起動する login プログラムの遷移設定を行なう。

暗号鍵とログイン情報を記録するために必要なアクセス権限を与える。

シェルを実行したときの遷移設定を行なう。デフォルトでは user_t にする。

pty (pseudo terminal) の再ラベル付けに使う SID の取得を許可する。

ssh が実行する login プログラムにもログインするのに必要な権限を与える。以下に実際の設定例を示す。

```
# domain、type の宣言
type sshd_t, domain, privuser, privrole,
                                privlog, auth, privowner;
type sshd_login_t, domain, privuser, privrole,
                                privlog, auth, privowner;
type sshd_exec_t, file_type, exec_type, sysadmfile;
type sshd_key_t, file_type, sysadmfile;
type sshd_tmp_t, file_type, sysadmfile, tmpfile;
type sshd_var_run_t, file_type, sysadmfile, pidfile;
file_type_auto_trans(sshd_t, var_run_t, sshd_var_run_t)
file_type_auto_trans(sshd_t, tmp_t, sshd_tmp_t)
# capabilities の許可
allow sshd_t self:capability { chown fowner fsetid
                                sys_tty_config dac_override net_bind_service
                                setuid setgid };

#init から fd を継承し使用する。
allow sshd_t init_t:fd inherit_fd_perms;
# ネットワーク利用許可
can_network(sshd_t)
#ptys の作成許可
can_create_pty(sshd)
# login プログラムの実行
domain_auto_trans(sshd_t, login_exec_t, sshd_login_t)
# shell の属性を得る
allow sshd_t shell_exec_t:file { getattr };
# key、lastlog,utmp,wtmp ファイルへのアクセス許可
allow sshd_t sshd_key_t:file rw_file_perms;
allow sshd_t var_log_t:file rw_file_perms;
allow sshd_t initrc_var_run_t:file rw_file_perms;
allow sshd_t wtmp_t:file rw_file_perms;
#domain user_t でシェルを実行
domain_auto_trans(sshd_t, shell_exec_t, user_t)
#sysadm_t でシェルを実行許可
domain_trans(sshd_t, shell_exec_t, sysadm_t)
```

```
# user domains.に signal 送信許可
allow sshd_t userdomain:process signal;
#pty の再ラベル付けに使う SID 取得のため
allow sshd_t security_t:security change_sid;
#sshd が作成した ptys の再ラベル付け。
allow sshd_t sshd_dev
pts_t:chr_file { relabelfrom relabelto };
# sshd_login_t のルール設定
# capabilities の利用許可
allow sshd_login_t self:capability { setuid setgid chown
                                fowner fsetid net_bind_service
                                sys_tty_config dac_override };

# ネットワーク利用許可
can_network(sshd_login_t)
# user_t でシェルを実行
domain_auto_trans(sshd_login_t, shell_exec_t, user_t)
#sysadm_t への遷移を許可
domain_trans(sshd_login_t, shell_exec_t, sysadm_t)
# pty の作成許可
allow sshd_login_t sshd_devpts_t:chr_file rw_file_perms;
#書き込み許可
allow sshd_login_t initrc_var_run_t:file rw_file_perms;
allow sshd_login_t wtmp_t:file rw_file_perms;
allow sshd_login_t var_log_t:file rw_file_perms;
# mail spool file の検索
allow sshd_login_t mail_spool_t:dir r_dir_perms;
allow sshd_login_t mail_spool_t:file getattr;
# ptys のラベル変更許可
allow sshd_login_t security_t:security change_sid;
# pts のラベル変更許可
allow sshd_login_t sshd_devpts_t:chr_file
                                { relabelfrom relabelto };
```

(3) RBAC の設定

先に `initrc_t` から `sshd_t` への遷移を許可したが、`initrc_t` の `role` でも `sshd_t` への遷移を許可しなければならない。`initrc_t` の `role` は、システムプロセスに付与される `system_r` である。`system_r` に `sshd_t` に遷移できるようにするため、`system_r` に `sshd_t` を加える。RBAC の設定ファイル `rbac` に以下を追加する。

```
role system_r types {  
    sshd_t  
}
```

(4) ファイルのラベル付け

TE の設定で宣言した type を、ファイルおよびディレクトリに対してラベル付けする。

```
/etc/ssh/primes          system_u:object_r:sshd_key_t  
/etc/ssh/ssh_host_key    system_u:object_r:sshd_key_t  
/etc/ssh/ssh_host_dsa_key system_u:object_r:sshd_key_t  
/etc/ssh/ssh_host_rsa_key system_u:object_r:sshd_key_t  
/usr/bin/ssh  
system_u:object_r:ssh_exec_t
```

2.1.7.7. FTP サーバ

FTP サーバ wu-ftpd の設定を取り上げる。使用した wu-ftpd はバージョン 2.6.1 である。ここではデフォルトの設定ファイルで用意されている policy/domain/system/ftpd.te ファイルを使って説明する。また、デフォルトの設定ファイルに対して匿名ユーザによるファイルのダウンロードの設定を追加している。

wu-ftpd のポリシー設定では新たに wu-ftpd のプロセスに付与する domain である ftpd_t を宣言する。wu-ftpd が使用するネットワークについても type を宣言する必要がある。また、FTP サーバにログインできるユーザを匿名ユーザのみとして、公開するファイルを /var/ftp/pub ディレクトリに置く。そして FTP サーバに対して、/var/ftp/pub ディレクトリ以下のファイルに対する読み込みのみ

を許可する設定をする。/var/ftp/pub ディレクトリおよびそれ以下のファイルに付与する type を ftpd_pub_file とする。

(1) ネットワークの設定

まず ftp サーバが使用するポートに付与する type を宣言する。ファイル types/network.te に以下を追加する。

```
type ftp_port_t, port_type;
```

次に、ftp_port_t と FTP サーバが使用する tcp21 番ポートを関連付けするためにファイル net_contexts に以下を追加する。

```
tcp 21 system_u:object_r:ftp_port_t
```

これによって、tcp21 番ポートを使用するには ftp_port_t へのアクセス許可が必要となる。

(2) TE の設定

TE の設定ではまず type の宣言を行なう。最初に FTP サーバのプロセスに付与する domain と、FTP サーバの実行ファイルに付与する type を宣言する。ftpd_t は、FTP サーバのプロセスに付与するので domain 属性を付与する。また、FTP サーバは syslogd を使ってログを出力するので privlog 属性も付与する。ftpd_exec_t は FTP サーバ実行ファイルに付与するので file_type 属性と exec_type 属性を付与する。また、管理者によってのみフルアクセスできるように sysadmfile も付与する。

次に ftpd_t がアクセスするファイルにラベル付けするオブジェクト type を定義する。また pid ファイルと FTP サーバログファイルの type をそれぞれ

れ宣言する。pid ファイルの type には、pid ファイルを示す pidfile 属性を付与する。それらのファイルが作成されたときに、type を遷移させるために file_type_auto_trans マクロを使う。また、ftpd のログ情報とログイン情報を記録するためのアクセス権限も付与する。

公開ファイルに付与する type である ftpd_pub_file の宣言と、公開ファイルに対しての読み込み許可を与える。

FTP サーバは、inetd または xinetd から起動されるので、これからファイルディスクリプタを引き継ぐ必要がある。

FTP サーバではいくつかの capability が必要なので、必要な capability を ftpd_t に付与する。

ftpd_t がネットワークを使用できるようにするため can_network マクロによって必要な権限を与える。

FTP でのデータ転送では FTP サーバ側からデータストリームの接続を行なうので can_tcp_connect マクロで接続許可を設定する。ここで inetd への接続許可としているが、まだパケットへのラベル付けがサポートされていないのでこのマクロは空文字列に置換されるだけである。

FTP サーバが終了したとき、inetd にシグナルを送るためその権限を与える。

FTP サーバで必要となる ls コマンドの実行許可を与える。ここで、bin_t へ実行許可を与えると bin_t が付与されているすべてのコマンドの実行許可を与えることになってしまうので、最低限のコマンド実行許可を与えるため個別に与えている。実際の設定例を以下に示す。

```
#domain, type の宣言
type ftpd_t, domain, privlog;
type ftpd_exec_t, file_type, sysadmfile, exec_type;
#pid ファイルの type と遷移の宣言
type ftpd_var_run_t, file_type, sysadmfile, pidfile;
file_type_auto_trans(ftp_t, var_run_t, ftpd_var_run_t)
#ftpd のログファイル xferlog の type 宣言と関連するアクセス権
type xferlog_t, file_type, sysadmfile;
file_type_auto_trans(ftp_t, var_log_t, xferlog_t)
allow ftp_t xferlog_t:file rw_dir_perms;
# ファイル wttmp にログイン情報を追加する権限
allow ftpd_t wttmp_t:file append;
#/var/log/lastlog ファイルにログイン情報を書き込むための権限
allow ftpd_t var_log_t:file write;
#/var/ftp/pub 以下に付与する type の宣言とアクセス許可
type ftpd_pub_file file_type,sysadmfile;
allow ftpd ftpd_pub_file:file r_file_perm;
allow ftpd ftpd_pub_file:dir r_dir_perm;
allow ftpd ftpd_pub_file:dir search;
#inetd プロセスから fd を引き継ぎ、使用する
allow ftpd_t inetd_t:fd inherit_fd_perms;
#inetd で開いたソケットを使う
allow ftpd_t inetd_t:tcp_socket rw_stream_socket_perms;
#FTP サーバに必要な capability を付与する
allow ftpd_t ftpd_t:capability { net_bind_service setuid
                               setgid fowner fsetid chown sys_resource };
# ネットワーク利用許可
can_network(ftp_t)
# inetd への接続許可。
can_tcp_connect(ftp_t,inetd_t)
#プロセスが終了したときに、inetd にシグナルを送る許可。
allow ftpd_t inetd_t:process sigchld;
# ls コマンドの実行許可。
can_exec(ftp_t, ls_exec_t)
```

FTP サーバは xinetd から起動されるので、xinetd が FTP サーバを実行したときに ftpd_t の domain に遷移させるようにする。また xinetd に ftpd が使用する 21 番ポートへのアクセスを許可する。xinetd に付与する domain は、inetd_t なので inetd_t の定義ファイル domain/system/inetd.te に以下を追加する。

```
domain_auto_trans(inetd_t, ftpd_exec_t, ftpd_t)
allow inetd_t ftp_port_t:tcp_socket name_bind;
```

(3) RBAC の設定

先に inetd_t から ftpd_t への遷移を許可したが、inetd_t の role でも ftpd_t への遷移を許可しなければならない。inetd_t の role は system_r である。system_r に ftpd_t に遷移できるようにするため system_r に ftpd_t を加える。RBAC の設定ファイル rbac に以下を追加する。

```
role system_r types {
    ftpd_t
}
```

(4) ファイルのラベル付け

TE の設定で宣言した type を、ファイルおよびディレクトリに対して再ラベル付けする。

```
/usr/sbin/in.ftpd          system_u:object_r:ftpd_exec_t
/var/ftp/pub(|/.**)       system_u:object_r:ftpd_pub_file_t
```

2.1.7.8. canna サーバ

canna サーバの設定を取り上げる。使用した canna サーバはバージョン 3.5b2 である。ここで説明する設定ファイルは、デフォルトで用意されていないので、新規に policy/domain/system/canna.te を作成した。

canna サーバのポリシー設定では、新たにサーバプロセス cannaserver に付与

する domain である `canna_t` を宣言し、`cannaserver` がアクセスする辞書ファイルと設定ファイルの `type` も宣言する。また、`cannaserver` が使用するネットワークについても `type` を宣言する必要がある。

(1) ネットワークの設定

まず `canna` サーバが使用するポートに付与する `type` を宣言する。ファイル `types/network.te` に以下を追加する。

```
type canna_port_t, port_type;
```

次に、`canna_port_t` と `canna` サーバが使用する `tcp5680` 番ポートを関連付けするために `net_contexts` ファイルに以下を追加する。

```
tcp 5680 system_u:object_r:canna_port_t
```

これによって `tcp5680` 番ポートを使用するには、`canna_port_t` へのアクセス許可が必要となる。

(2) TE の設定

TE の設定ではまず `type` 宣言をする。`cannaserver` プロセスに付与する domain と `canna` の実行ファイルに付与する `type` を宣言する。`canna_t` は、`canna` サーバの実行プロセスに付与するので domain 属性を付与する。また `canna` サーバは `syslogd` を使ってログを出力するので、それを許可する `privlog` 属性も付与する。`canna_exec_t` は `cannaserver` 実行ファイルに付与するので `file_type` 属性と `exec_type` 属性を付与する。また、管理者によってのみフルアクセスできるように `sysadmfile` も付与する。

次に、`canna_t` がアクセスするファイルにラベル付けするオブジェクト

type を宣言する。canna サーバの辞書ファイルと設定ファイルに付与する type を宣言してアクセス権限を設定する。

canna サーバが作成する各ユーザの辞書ファイルに付与する type を宣言してアクセス権限を設定する。

canna サーバにいくつかの capability が必要なので、必要な capability を canna_t に付与する。

canna サーバは、init から起動されるので init からファイルディスクリプタを引き継ぐ必要がある。

canna_t がネットワークを使用できるようにするため、can_network マクロによって必要な権限を与える。canna サーバが使用するポートに TCP ソケットをバインドしなければならないので、ポートへのバインドを許可する。

kinput2 などの日本語入力プログラムと UNIX ドメインで通信するために /tmp/.iroha_unix ディレクトリ以下に付与する type の宣言とアクセス権限を与える。

canna サーバが起動したときに、前に UNIX ドメイン通信で使用していたファイルを削除する権限を与える。

canna サーバを管理者によって起動したい場合は管理者の domain canna_t からの遷移を設定する。

```
#canna の domain, type の宣言
type canna_t, domain, privlog;
type canna_exec_t, file_type, sysadmfile, exec_type;
#/var/lib/canna に付与する type とアクセス権限の設定
type var_lib_canna_t ,file_type,sysadmfile;
allow canna_t var_lib_canna_t:file rw_file_perms;
allow canna_t var_lib_canna_t:file create_file_perms;
allow canna_t var_lib_canna_t:dir create_dir_perms;
allow user_t var_lib_canna_t:file r_file_perms;
allow user_t var_lib_canna_t:dir r_file_perms;
```

```
allow user_t var_lib_canna_t:dir search;
#ユーザの辞書ファイルに付与する type とアクセス権限の設定
type user_canna_dic_t ,file_type, sysadmfile;
allow canna_t user_canna_dic_t:file rw_file_perms;
allow canna_t user_canna_dic_t:file create_file_perms;
allow canna_t user_canna_dic_t:dir create_dir_perms;
allow user_t user_canna_dic_t:file rw_file_perms;
allow user_t user_canna_dic_t:file create_file_perms;
allow user_t user_canna_dic_t:dir create_dir_perms;
#必要な capability を付与する。
allow canna_t canna_t:capability { setuid setgid };
init プロセスから fd を引き継ぎ、使用する。
allow canna_t init_t:fd inherit_fd_perms;
# ネットワーク利用許可
can_network(canna_t)
can_unix_connect(canna_t,user_t)
allow canna_t canna_port_t:tcp_socket name_bind;
#UNIX ドメインソケットで使う/tmp/.iroha_unix に付与する type 宣言
とアクセス権限の設定
type canna_tmp_t, file_type, sysadmfile, tmpfile;
allow canna_t canna_tmp_t:sock_file rw_file_perms;
allow canna_t canna_tmp_t:sock_file create_file_perms;
allow canna_t canna_tmp_t:dir rw_file_perms;
allow canna_t canna_tmp_t:dir {add_name search};
allow canna_t canna_tmp_t:unix_stream_socket {name_bind};
allow user_t canna_tmp_t:dir {search };
allow user_t canna_tmp_t:unix_stream_socket {name_bind };
allow user_t canna_tmp_t:sock_file rw_file_perms;
#canna が立ち上がるときに /tmp/.iroha_UNIX を削除
allow initrc_t canna_tmp_t:sock_file unlink;
allow initrc_t canna_tmp_t:dir remove_name;
allow initrc_t canna_tmp_t:dir rw_file_perms;
#管理者が起動させたいとき
domain_auto_trans(sysadm_t, canna_exec_t,canna_t)
```

canna サーバは rc スクリプトから起動されるので、init が実行したときに canna_t に遷移させるようにする。rc スクリプトに付与する domain は、initrc_t なので canna サーバを実行したときに canna_t に遷移させるように定義する。initrc_t の定義は、domain/system/initrc.te なので以下を追加する。

```
domain_auto_trans(initrc_t,canna_exec_t,canna_t)
```

(3) RBAC の設定

先に `initrc_t` から `canna_t` への遷移を許可したが、`initrc_t` の `role` でも `canna_t` への遷移を許可しなければならない。`initrc_t` の `role` はシステムプロセスの `role system_r` である。`system_r` に `canna_t` に遷移できるようにするため `system_r` に `canna_t` を加える。RBAC の設定ファイル `rbac` に以下を追加する。また、管理者によって起動させたいときは管理者 `domain` に `sysadm_t` にも追加する。

```
role system_r types {
    |
    |   canna_t
    |   |
    |   }
    | # 管理者が起動させるとき。
    | role sysadm_r types {
    |   |
    |   |   canna_t
    |   |   |
    |   |   }
    | }
}
```

(4) ファイルのラベル付け

TE の設定で宣言した `type` を、ファイルおよびディレクトリに対して再ラベル付けする。

```
/usr/sbin/cannakill      system_u:object_r:canna_exec_t
/usr/sbin/cannaserver   system_u:object_r:canna_exec_t
/usr/bin/cannacheck     system_u:object_r:canna_exec_t
/usr/bin/cannastat      system_u:object_r:canna_exec_t
/usr/bin/catdic          system_u:object_r:canna_exec_t
/var/lib/canna(|/.*)    system_u:object_r:var_lib_canna_t
/var/lib/canna/dic/user(|/.*)
                        system_u:object_r:user_canna_dic_t
/tmp/\.iroha\_unix(|/.*) system_u:object_r:canna_tmp_t
```

3. 付録

3.1. デフォルトマクロ一覧

表 3.1-1 マクロ一覧

マクロ分類	適用先	マクロ名	説明
クラス グルーピング マクロ	ディレクトリ および ファイル	dir_file_class_set	ディレクトリおよび、 全ファイル関連クラス を表す。
		file_class_set	全ファイル関連クラス を表す。
		notdev_class_set	デバイスファイル以外 の全ファイル関連クラ スを表す。
		devfile_class_set	デバイスファイル関連 クラスを表す。
	ソケット	socket_class_set	全ソケット関連クラス を表す。
パーミッション グルーピング マクロ	ファイル	stat_file_perms	ファイルに対して stat および、access コマンドの実行を許可 するパーミッションを 表す。
		x_file_perms	ファイルの実行を許可 するパーミッションを 表す。
		r_file_perms	ファイルの読み込みを 許可するパーミッシ ョンを表す。
		rx_file_perms	ファイルの読み込みお よび、実行を許可す るパーミッションを表 す。
		rw_file_perms	ファイルの読み込みお よび、書き込みを許 可するパーミッション を表す。
		link_file_perm	ファイルのリンク、ア ンリンクおよびリネー ムを許可するパーミ ッションを表す。

マクロ分類	適用先	マクロ名	説明
		create_file_perms	ファイルの作成、読み込み、書き込み、リンク、リネームおよびアンリンクを許可するパーミッションを表す。
	ディレクトリ	r_dir_perms	ディレクトリの読み取りを許可するパーミッションを表す。
		rw_dir_perms	ディレクトリの読み取りおよび書き込みを許可するパーミッションを表す。
		create_dir_perms	ディレクトリの作成を許可するパーミッションを表す。
	データグラム および raw ソケット	rw_socket_perms	ソケットに対する読み込みおよび書き込みを許可するパーミッションを表す。
		create_socket_perms	ソケットの作成、読み込みおよび書き込みを許可するパーミッションを表す。
	ストリーム ソケット	rw_stream_socket_perms	ソケットの読み込みおよび書き込みを許可するパーミッションを表す。
		create_stream_socket_perms	ソケットの作成、読み込みおよび書き込みを許可するパーミッションを表す。
	ファイルディスクリプタ	inherit_fd_perms	ファイルディスクリプタの引継ぎおよび利用を許可するパーミッションを表す。
		receive_fd_perms	ローカルソケットによるプロセス間通信において、ファイルディスクリプタを受信し、それを利用することを許可するパーミッションを表す。
	ファイルシステム	mount_fs_perms	ファイルシステムのマウントおよびアンマウントを許可するパーミッションを表す。

マクロ分類	適用先	マクロ名	説明
	シグナル	signal_perms	任意のシグナルの送信を許可するパーミッションを表す。
	パケット	packet_perms	ネットワークパケットの送受信を許可するパーミッションを表す。
ルールマクロ	domain の遷移	domain_trans	プログラムの domain 遷移ルールを定義するマクロ。
		domain_auto_perms	domain_trans マクロに追加ルールを定義するマクロ。実行プログラムの domain 遷移の自動化を定義する。
	ファイル type の遷移	file_type_trans	ファイルの type 遷移ルールを定義するマクロ。
		ffile_type_auto_trans	file_type_trans マクロに追加ルールを定義するマクロ。ファイルの type 遷移の自動化を定義する。
	プログラムの実行	uses_shlib	ダイナミックローダを実行し、共有ライブラリからのコード実行を許可するルールを定義する。
		can_exec	domain 一般に対して domain 遷移を行うこと無くプログラムを実行することを許可するルールを定義する。
		can_exec_any	domain 一般に対してあらゆるシステムプログラムを実行することを許可するルールを定義する。
	ネットワーク通信	can_network	UDP あるいは TCP ソケット経由でネットワーク通信を許可するルールを定義する。
	ネットワーク TCP 通信	can_tcp_connect	TCP 接続を許可する domain のペアを定義する。

マクロ分類	適用先	マクロ名	説明
		can_udp_send	TCP ソケットを利用した通信を許可する domain のペアを定義する。
	ネットワーク UDP 通信	can_unix_connect	UDP 接続を許可する domain のペアを定義する。
		can_unix_send	UDP ソケットを利用した通信を許可する domain のペアを定義する。
	システム・パラ メータ操作	can_sysctl	あらゆるシステムパラメータ（カーネルパラメータ）の修正を domain に対して許可するルールを定義する。
	仮想端末	can_create_pty	一般ユーザの domain あるいは管理ユーザ domain に対して、仮想端末の作成を許可し同端末でのアクセスを許可するルールを定義する。
		can_create_other_pty	別の domain に代って仮想端末を作成し、同端末でのアクセスを domain に対して許可するルールを定義する。

3.2. デフォルト属性一覧

表 3.2-1 ファイル security.te

type 名	属性	説明
security_t	属性なし	セキュリティサーバに関連するシステムコールの利用権限を与える type。
policy_config_t	file_type,sysadmfile	コンパイル済みのポリシーコンフィギュレーションファイル (/ss_policy) へのアクセス権を与える type。新たなポリシーを OS に読み込む権限もこの type に依存する。
policy_src_t	file_type,sysadmfile	ポリシーコンフィギュレーションのソースファイルに対するアクセス権を与える type。
file_labels_t	file_type,sysadmfile	ファイルシステムごとに管理されている、ラベルのマッピング情報へのアクセス権を与える type。
no_access_t	file_type,sysadmfile	管理ユーザにのみアクセスを許可したいファイルに付与する type。

表 3.2-2 ファイル device_t.te

type 名	属性	説明
device_t	file_type	デバイスファイルを含むディレクトリに対して付与される type。同ディレクトリ内のファイルに対してもデフォルトではこの type が付与される。
null_device_t	file_type	null デバイスに対して付与される type。
zero_device_t	file_type	zero デバイスに対する type。
console_device_t	file_type	コンソールデバイスへ付与される type。
memory_device_t	file_type	メモリデバイスの /dev/kmem、/dev/mem と /dev/port への raw アクセスを制限する type。
random_device_t	file_type	乱数を取得する際に利用するデバイスへ付与する type。
devtty_t	file_type	/dev/tty に付与される type。
tty_device_t	file_type	tty デバイスに対して初期化時に付与される type。

type 名	属性	説明
fixed_disk_device_t	file_type	固定ディスクデバイスへ付与される type。固定ディスクデバイスへの raw アクセスを制御する。
removable_device_t	file_type	リムーバブルデバイスへ付与される type。リムーバブルデバイスへの raw アクセスを制御する。
clock_device_t	file_type	クロックデバイスへ付与される type。リアルタイムクロックへのアクセスを制御する。
misc_device_t	file_type	適切な制御形態が十分に調査されていないデバイスに付与される type。
psaux_t	file_type	マウスデバイスである /dev/psaux へ付与される type。
mouse_devicet	file_type	マウスデバイス /dev/psaux、 /dev/*mouse* や /dev/input/*mouse* に付与される。
agp_device_t	file_type	主メモリの一部をグラフィックメモリとして使うグラフィックカードを使用するためのデバイス /dev/agpgart に付与される type。
dri_device_t	file_type	DRI (Direct Rendering Interface) を使用するためのデバイス /dev/dri 以下のファイルに付与される type。
sound_device_t	file_type	サウンドデバイス /dev/mixer、 /dev/dsp、 /dev/audio や /dev/midi に付与される type。

表 3.2-3 ファイル file.te

type 名	属性	説明
unlabeled_t	fs_type, root_dir_type	ラベル付けがまだサポートされていないファイルに付与される type。同ファイルに対するアクセスを制御する。
fs_t	fs_type	レベル付けされていないファイルシステムに付与される type。
file_t	file_type, root_dir_type, sysadmfile	ディレクトリおよびファイルに付与されるデフォルトの type。ディレクトリおよびファイルに対するアクセスを制御する。

type 名	属性	説明
root_t	file_type,sysadmfile	root ディレクトリに付与される type。root ディレクトリに対するアクセスを制御する。全 domain がこの type に対する読み込みを許可される。
lost_fount_t	file_type,sysadmfile	lost+found ディレクトリおよびその中のファイルに対して付与される type。同ディレクトリおよびファイルに対するアクセスを制御する。ファイルシステムの管理プログラムおよび管理ユーザ domain のみがこの type に対するパーミッションを与えられる。
boot_t	file_type,root_dir_type,sysadmfile	ブートディレクトリおよび同ディレクトリ内のファイルに対して付与される type。上記ディレクトリおよびファイルに対するアクセスを制御する。すべての domain はこの type に対して読み込みを許可される。
boot_runtime_t	file_type,sysadmfile	/boot/kernel.h に付与される type。このファイルはシステム初期化時に作成されるため boot_t とは区別された本 type が付与される。すべての domain はこの type に対して読み込みを許可される。
tmp_t	file_type,sysadmfile,tmpfile	テンポラリディレクトリに付与される type。テンポラリディレクトリに対するアクセスを制御する。
etc_t	file_type,sysadmfile	/etc ディレクトリに付与される type。システムのコンフィギュレーション情報へのアクセスを制御する。
ld_so_cache_t	file_type,sysadmfile	共有ファイル検索用キャッシュファイルに付与される type。/etc/ld.so.cache ファイルに付与される。
etc_runtim_t	file_type,sysadmfile	システム初期化処理の間に作成される設定ファイルに付与される type。/etc/HOSTNAME や/etc/mtab ファイルなどに付与される。
etc_aliases_t	file_type,sysadmfile	sendmail の別名定義機能で利用されるエイリアスデータベース /etc/aliases.db ファイルに付与される type。「sendmail_t」domain はこのファイルに対して読み取りおよび書き込みが許可される。

type 名	属性	説明
etc_mail_t	file_type,sysadmfile	/etc/mail ディレクトリに付与される type。「sendmail_t」domainはこのディレクトリに対して読み取りおよび書き込みが許可される。
resolv_conf_t	file_type,sysadmfile	名前解決設定ファイルのために定義された type。/etc/resolv.conf に付与される。すべての domain に読み込み許可が与えられている。
etc_mrtg_t	file_type,sysadmfile	ネットワーク負荷監視ツール MRTG の設定ファイルのために定義された type。/etc/mrtg ディレクトリおよびその中のファイルに付与される。system_cron_tdomain のみに許可が与えられている。
lib_t	file_type,sysadmfile	システムのライブラリに付与される type。すべての domain が本 type に対して読み取りが許可される。本 type の修正は管理ユーザの domain に対してのみ許可される。
shlib_t	file_type,sysadmfile	システムの共有ライブラリに付与される type。すべての domain が本 type に対して読み取りを許可されている。
ld_so_t	file_type,sysadmfile	システムのダイナミックロードに付与される type。すべての domain は本 type に対する読み取りを許可されている。
bin_t	file_type,sysadmfile	システムバイナリに付与される type。すべての domain は本 type の読み取りを許可されており、いくつかの domain が本 type の実行を許可されている。
ls_exec_t	file_type,exec_type,sysadmfile	ls コマンドに付与される type。
chpasswd_exec_t	file_type,exec_type,sysadmfile	/sbin/pwdb_chkpwd と /sbin/unix_chkpwd に付与される type。
sbin_t	file_type,sysadmfile	システムバイナリに付与される type。
usr_t	file_type,root_dir_type,sysadmfile	/usr ディレクトリに付与される type。
src_t	file_type,sysadmfile	システム実行プログラムのソースの type。/usr/src などに付与されるすべての domain に読み込み権限が与えられている。
var_t	file_type,root_dir_type,sysadmfile	/var ディレクトリに付与される type。

type 名	属性	説明
var_run_t	file_type,sysadmfile	/var/run ディレクトリに付与される type。
var_log_t	file_type,sysadmfile	/var/log ディレクトリに付与される type。
var_lock_t	file_type,sysadmfile	/var/lock ディレクトリに付与される type。
var_lib_t	file_type,sysadmfile	/var/lib ディレクトリに付与される type。
var_spool_t	file_type,sysadmfile	/var/spool ディレクトリに付与される type。
var_yp_t	file_type,sysadmfile	/var/yp ディレクトリに付与される type。本 type を修正可能なのは、管理ユーザの domain および、「ypbind_t」 domain である。
var_log_sa_t var_log_ksyms_t var_lib_nfs_t var_lib_rpm_t	file_type,sysadmfile	/var ディレクトリにあるいくつかのディレクトリのための type
wtmp_t	file_type,sysadmfile	/var/log/wtmp ディレクトリに付与される type。
catman_t	file_type,sysadmfile	/var/catman ディレクトリに付与される type。
at_spool_t	file_type,sysadmfile	/var/spool/at ディレクトリに付与される type。
cron_spool_t	file_type,sysadmfile	/var/spool/cron ディレクトリに付与される type。
lpd_spool_t	file_type,sysadmfile	/var/spool/lpd ディレクトリに付与される type。
mail_spool_t	file_type,sysadmfile	/var/spool/mail ディレクトリに付与される type。
mqueue_spool_t	file_type,sysadmfile	/var/spool/mqueue ディレクトリに付与される type。
man_t	file_type,sysadmfile	マニュアルページのディレクトリおよび、ファイルへ付与される type。
readable_t	file_type,sysadmfile	この type が付与されたファイルは、すべての domain に読み込み権限が許可される。
writable_t	file_type,sysadmfile	この type が付与されたファイルは、すべての domain に読み書き権限が許可される。
poly_t	file_type,sysadmfile	polyinstantiated ディレクトリのために定義された type。
swapfile_t	file_type,sysadmfile	スワップファイルのために定義された type。
tmpfs_t	file_type,sysadmfile,fs_type,root_dir_type	tmpfs のために定義された type。

type 名	属性	説明
devfs_t	fs_type, root_dir_type	devfs のために定義された type。

表 3.2-4 ファイル devpts.te

type 名	属性	説明
ptmx_t	file_type, sysadmfile	擬似端末のマスタデバイス /dev/ptmx に付与される type。
devpts_t	fs_type, root_dir_type	/dev/pts ディレクトリに付与される type。

表 3.2-5 ファイル network.te

type 名	属性	説明
any_socket_t	socket_type	UDP あるいは、raw IP によるやりとりの際のあて先ソケットに対してデフォルトで付与される type。ネットワークを利用する全 domain が本 type を付与されたソケットへの送信を許可されている (can_network マクロによりパーミッションが与えられる)。
icmp_socket	socket_type	ICMP メッセージを送信する際に利用するカーネルソケットに付与する type。本 type が付与されるソケットでは、raw IP メッセージの送受信が許可される。本 type が付与されるソケットは、カーネルが内部的に利用するソケットであるため、domain に対しては一切パーミッションは与えられない。
tcp_socket_t	socket_type	TCP リセットパケットを送信する際に利用するカーネルソケットに付与する type。本 type が付与されたソケットでは、TCP メッセージの送受信が許可される。本 type が付与されるソケットは、カーネルが内部的に利用するソケットであるため、domain に対しては一切パーミッションは与えられない。
port_t	port_type	INET のポート番号に付与するデフォルトの type。すべての domain が本 type を付与されたポート番号に対してバインドを許可される。いくつかのポート番号には特有の type が定義される。

type 名	属性	説明
ftp_port_t	port_type	ftpd が利用するポート番号に付与する type。
telnet_port_t	port_type	telnet が利用するポート番号に付与する type。
smtp_port_t	port_type	smtp が利用するポート番号に付与する type。
http_port_t	port_type	httpd が利用するポート番号に付与する type。
rlogin_port_t	port_type	rlogin が利用するポート番号に付与する type。
rsh_port_t	port_type	rsh が利用するポート番号に付与する type。
printer_port_t	port_type	lpd が利用するポート番号に付与する type。
named_port_t	port_type	named が利用するポート番号に付与する type。
netif_t	netif_type	ネットワークインタフェースに対して付与するデフォルトの type。
netif_eth0_t netif_eth1_t netif_lo_t	netif_type	ネットワークインタフェースに対して、インタフェース名を考慮して付与する type。いくつかの domain および管理ユーザの domain は、すべてのネットワークインタフェースのコンフィギュレーションを許可され、すべての domain はあらゆるネットワークインタフェースにおいて送受信することを許可される。
netmsg_t netmsg_eth1_t netmsg_lo_t	netmsg_type	ネットワークインタフェースにて受信されるメッセージ(ラベル付けされていないメッセージ)に対して付与するデフォルトの type。
node_t	node_type	ノードに対して付与するデフォルトの type。すべての domain は、あらゆるノード type に対してメッセージを送信する許可が与えられる。
node_lo_t	node_type	ループバックアドレスに対して付与するデフォルトの type。
node_internal_t	node_type	ネットワークインタフェースが受信した、ラベル付けされていないメッセージに付与されるデフォルトの type。

表 3.2-6 ファイル nfs.te

type 名	属性	説明
nfs_t	fs_type, root_dir_type	NFS のファイルシステムおよび、そのファイルに付与されるデフォルトの type。すべての domain は、本 type に対して読み取りおよび書き込みが許可されている。

表 3.2-7 ファイル procfs.te

type 名	属性	説明
proc_t	fs_type, root_dir_type	/proc ディレクトリおよび同ディレクトリ下のファイルに付与される type。全 domain が本 type に対する読み取りを許可される。
proc_kmsg_t	属性なし	/proc/kmsg ファイルに付与される type。「klog」プログラムに付与された domain のみが、本 type に対する読み取りを許可される。
proc_kcore_t	属性なし	/proc/kcore ファイルに付与される type。現在、本 type に対して読み取りを許可された domain は存在しない。
sysctl_t	属性なし	/proc/sys ディレクトリおよび同ディレクトリ下のファイルに付与される type。
sysctl_fs_t	属性なし	/proc/sys/fs ディレクトリおよび同ディレクトリ下のファイルに付与される type。
sysctl_kernel_t	属性なし	/proc/sys/kernel ディレクトリおよび同ディレクトリ下のファイルに付与される type。
sysctl_modprobe_t	属性なし	/proc/sys/kernel/modprobe ファイルに付与される type。
sysctl_net_t	属性なし	/proc/sys/net ディレクトリおよび同ディレクトリ下のファイルに付与される type。

type 名	属性	説明
sysctl_net_unix_t	属性なし	/proc/sys/net/unix ディレクトリに付与される type。
sysctl_vm_t	属性なし	/proc/sys/vm ディレクトリおよび同ディレクトリ下のファイルに付与される type。
sysctl_dev_t	属性なし	/proc/sys/dev ディレクトリおよび同ディレクトリ下のファイルに付与される type。