タイプ継承を可能にする SELinux ポリシーコンパイラの拡張

海外 浩平*

*日本電気株式会社 Linux 推進センター／kaigai@ak.jp.nec.com

概要

Linux オペレーティングシステムのセキュリティ機能を強化する SELinux は、各種のディストリビューションでも標準採用されるなど、本格的な普及が始まろうとしている。

SELinux はセキュリティポリシーに基づいて一元的にアクセス制御を行う点に特徴があり、従来の UNIX セマンティクスに基づいたアクセス制御と比べ、非常に詳細なアクセス制御を行うことができる。その反面、アクセス制御を記述するセキュリティポリシーの記述が複雑になるという問題を抱えている。

著者は SELinux のセキュリティポリシー・コンパイラ(以下 checkpolicy)を拡張し、従来バージョンとの完全な互換性を維持したままで、アクセス制御の最小単位であるタイプに継承の概念を追加した。これにより、定義済みタイプを利用して冗長性を排除したセキュリティポリシーの記述を行うことが可能になり、ポリシーの一貫性・可読性の向上を実現する。

本論文では、checkpolicy に対して行った拡張と追加した文法、およびこれを利用したセキュリティポリシーを記述するための方法について述べる。

1. はじめに

Linux2.6.x 系列のカーネル[1]は、従来の UNIX セマンティクスに基づいたアクセス制御に加えて、セキュリティ強化のためのフレームワークである LSM[1]がマージされ、それを利用したセキュリティ強化モジュールである SELinux は Linux の OS レベルセキュリティ強化の切り札として注目を集めている。

2000 年に米国国家安全保障局(NSA)によって公開された SELinux[2]は、TE[2]/RBAC[3] と呼ばれるアクセス制御機能をシステム全体に適用することでプロセスの権限を必要最小限に抑制することが可能[3][4]で、仮にプログラムに何らかの脆弱性が存在した場合にも不必要なリソースへのアクセスを禁止することでシステム全体へ脆弱性の影響が拡散することを抑制する。

SELinux におけるアクセス制御は全てセキュリティポリシー[5]と呼ばれるファイルに記述されたルールに基づいて実施され、明示的に許可されたルール以外は実行を拒否する。

SELinux は OS の管理下にあるリソースを抽象化するために、タイプと呼ばれる識別子を付与する。セキュリティポリシーはタイプ間の関係を定義することで記述され、システムコール呼び出し側プロセスのタイプ(Subject)と、被参照側リソースのタイプ(Object)の間に許可されるアクションを逐一記述することによって行われる。

SELinux のアクセス制御ルールは、既存の UNIX セマンティクスに比べて細部にわたるアクセス制御を行うことができる。だが一方で、設定項目が膨大かつ多岐にわたり非常に難解であるために、独自のセキュリティポリシーを設定することが困難であるという問題を抱えている[6]。

特に、これまでは複数のタイプをグループ化して扱う機能が不十分であったため、類似のタイプに対して同一の権限を複数回割り当てなければならず、冗長な記述が必要であった。

これに対し、本論文で紹介する checkpolicy の拡張は、タイプ間に親子関係を定義してアクセス権限を継承することを可能にする。これにより、類似のアクセス権限を一括して付与することが可能になり、セキュリティポリシー記述の冗長性を大幅に削減できる。

その結果、セキュリティポリシーの一貫性や可読性が向上し、ヒューマンエラーを減少させることができる。

---

[1] LSM: Linux Security Module
[2] TE: Type Enforcement
[3] RBAC: Role Based Access Control

2. SELinux の Type Enforcement

SELinux におけるアクセス制御は TE と RBAC という２つの機能によって実現され、それを MAC[4]に基づいて全てのユーザ/プロセスに強制する。

SELinux の TE 機能は、プロセスやファイル・ディレクトリ・ソケット・共有メモリなどの OS の管理下にあるオブジェクトに対して、それぞれに一つだけタイプと呼ばれる SELinux 固有の識別子を付与する。

これによって、各オブジェクトは OS 上での実装方法に関係なくセキュリティ属性のみがタイプという形で抽象化されることとなり、セキュリティポリシーの記述はこれら属性同士の関係の定義という形に単純化される。

このうち、プロセスに対して付与されたタイプのことを特にドメインと呼ぶ。

SELinux のアクセス許可は、これらタイプ間にどういったアクションが許可されるかということを列挙することで記述される。タイプ間のアクションが明示的に許可されなければ、全て拒否アクションとして扱われる。

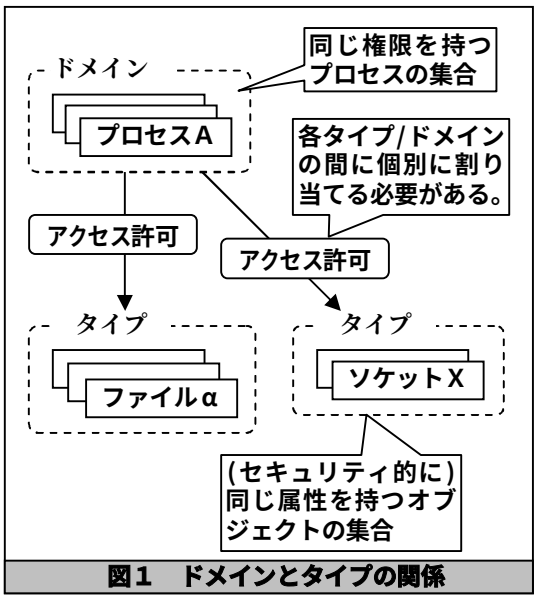

図１　ドメインとタイプの関係

図１に示すように、TE はプロセスが fork() されてから exit()するまで、どのような範囲のオブジェクトに、どのような種類のアクションを行うかということに注目した概念で、ドメイン内で動作するプロセスの振る舞いをよく理解した上で設定を行う必要がある。

同一のタイプで識別されるオブジェクトには全く同一の種類のアクセス権限が割り当てられる。少しでも異なるアクセス権限が必要な場合は、新たに別のタイプを作成して権限を割り当て直さなければならない。

これらのアクセス許可は、設定ファイルの記述を checkpolicy コマンドが解釈することでカーネルにロード可能なバイナリ形式にコンパイルされる。図２に、SELinux のアクセス許可構文の一例を示す。

```
allow ftpd_t ftpd_anon_t:file {read};
⇒ FTP サーバが anonymous FTP 用のファイルに Read アクセスを可能にする。

allow user_t self:process {signal};
⇒ ユーザプロセスが同一ドメインで動作するプロセスにシグナル送出を許可

allow <Subject> <Object>:<Class> <perms>;
  <Subject> : プロセスのドメインを指定
  <Object>  : オブジェクトのタイプを指定
  <Class>   : 権限のクラス(種別)を指定
  <perms>   : 個別パーミッションを指定
```

図２　アクセス許可の記述法

セキュリティポリシーを記述するためには、以上のようなアクセス許可構文を必要なだけ記述しなければならない。

典型的なアクセス許可パターンに関しては、NSA の管理する標準ポリシーに含まれる定義済みマクロによって記述することも可能である[7]。

しかし、ユーザの求めるポリシー設定が定義済みマクロと適合しない場合には、スクラッチからアクセス許可設定を記述しなければならない。細分化されたアクセス許可を列挙することは記述性に劣るほか、類似のタイプと同じ権限を付与し直すことは冗長である。また、アクセス許可の記述にはプロセスの挙動とセキュリティポリシー、定義済みマクロに対する高度な知識が要求される。

そのため、SELinux のセキュリティポリシーを独自にカスタマイズすることは難易度の高い作業であると考えられている。

[4] MAC : Mandatory Access Control

3. 継承に基づくタイプ定義

前節で述べたポリシー設定の困難さは、主に異なるタイプ間の関係を定義する方法が不十分であるために、類似のタイプ間で共通の権限を全て個別に付与しなければならないことに起因している。これによりセキュリティポリシーの記述は冗長なものとなって可読性に悪影響を与え、ヒューマンエラーの入り込む余地を大きくする。

このような問題を解決するため、SELinux のポリシー記述文法に"タイプ継承"の概念を取り入れ、複数の異なるタイプ間に親子関係を定義することを可能にした。このモデルの下では、明示的に子タイプに権限を継承するように指示されたアクセス許可は指定したタイプとその子孫のタイプに継承され、共通の属性に対して重複したアクセス許可を付与する必要がなくなる。

例として、図３のように FTP サーバ/HTTP サーバがユーザのホームディレクトリにアクセスする典型的な例を考えることとする。

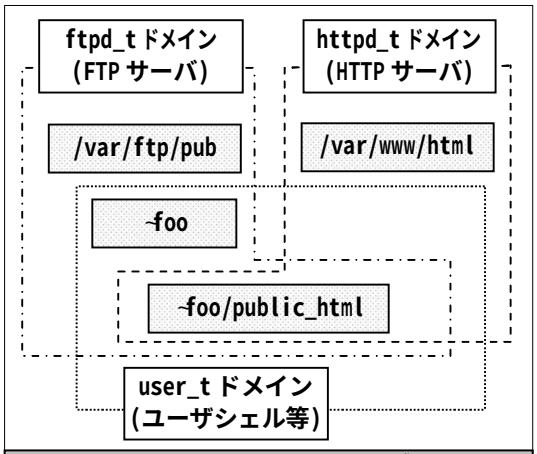

図３ シェル/FTP/HTTP の例

~foo 以下の全てのファイルには、ユーザシェル/FTP サーバがアクセスする可能性があり、~foo/public_html 以下にはそれに加えて HTTP サーバがアクセスする可能性がある。

FTP/HTTP サーバは /var 以下にそれぞれのデータ用ディレクトリを持っており、~foo と ~foo/public_html に対しても最低限これと同様のアクセス権を付与し直さなければならない。

つまり、従来のセキュリティポリシー記述においては、~foo 以下に FTP サーバ/シェルの設定を行い、~foo/public_html 以下には HTTP/FTP サーバとシェルに対するアクセス設定が個別に必要となる。

また、図４のように新たに Samba をインストールしてユーザのホームにアクセス許可を行う場合、~foo と~foo/public_html には別のタイプが付与されているので、これらの両方についてそれぞれ Samba サーバ向けの設定を行う必要がある。

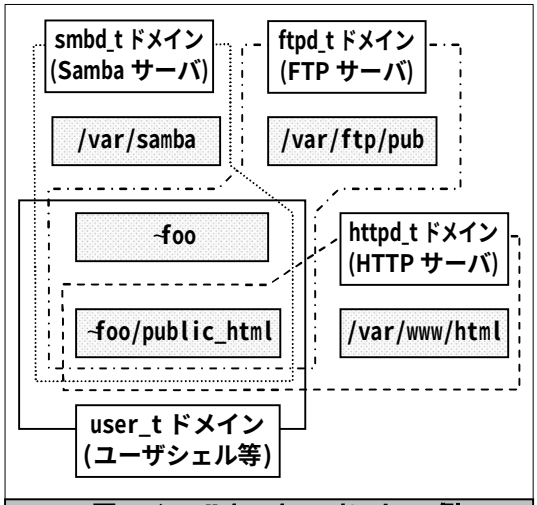

図４ シェル/FTP/HTTP/Samba の例

だが、これは二度手間になるだけでなく、設定漏れや設定ミスなどのヒューマンエラーを誘発する要因にもなる。

上記の例を、タイプ間の継承関係に基づくモデルを用いて図５のように簡潔に記述することができる。

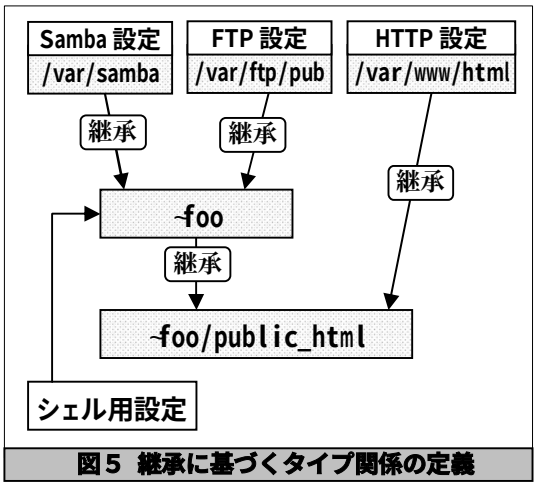

図５ 継承に基づくタイプ関係の定義

図５の例では、~foo はユーザシェル用の設定に加えて FTP/Samba サーバ用のアクセス許可設定を継承し、加えて~foo/public_html はそれら全ての設定と HTTP サーバ用のアクセス許可設定を継承している。

これは、~foo に対してアクセスを許可されたユーザシェルが、異なるタイプを付与された ~foo/public_html に対しても同一のアクセス権を保持していることを意味する。

HTTP サーバや FTP サーバ、Samba も同様に、各々がデフォルトでサービス対象としているディレクトリに対する設定を、~foo や ~foo/public_html に引き継がせて使用することができる。

継承に基づくタイプ間の親子関係というモデルを導入することにより、これまで各タイプに対して個別に全てのアクセス権限を割り当てなければならなかったところが、複数のタイプ間で共有可能なアクセス権限を親タイプ側で一括して付与することが可能となる。

また、タイプを最初から定義するためには、約束事として決められている様々なアクセス権の付与が必要になる場合も少なくないが、タイプ継承の概念を導入することで、ユーザは類似タイプとの差分に注目するだけで済む。

タイプ継承の類似機能として、従来から checkpolicy には複数のタイプに一度に権限を付与するため属性(attribute)がサポートされている。属性に対して権限を付与すると、属性がアタッチされたタイプ全てに権限が付与されるという性質があり、継承されたタイプに権限を付与するタイプ継承のアプローチに類似している。

しかし、属性は継承をサポートしないため、類似のアクセスパターンを持つタイプに対して全ての属性をアタッチし直さなければならず、セキュリティポリシーの記述が冗長になるという問題は解決できない。

タイプの継承では、アタッチされた属性も含めて子タイプに引き継がれるので、このような問題は解決可能である。

以上のタイプ継承モデルを checkpolicy に実装した。拡張された checkpolicy の詳細を次章で述べる。

4. checkpolicy への追加機能・実装

4-1 checkpolicy の追加機能

継承概念に基づくタイプ定義を実現するために、SELinux のセキュリティポリシ・コンパイラ checkpolicy に以下の拡張を加えた。

(1) TYPE ... EXTENDS ～ ; 構文の追加
(2) TYPEEXTENDS 構文の追加
(3) @プレフィクスの追加

(1)の TYPE ... EXTENDS～;構文は、従来のタイプ定義構文のオプショナルな拡張として提供され、図６に示す構文を持つ。

新しくタイプを宣言する際に、EXTENDS より後ろで指定した親タイプ/属性を継承したタイプを作成することができる。上記の例では、**ftp_anon_t** タイプを継承したタイプである **user_ftp_t** を定義し、**ftp_anon_t** に対して付与される権限を引き継いでいる。

```
TYPE <タイプ名>
  [ALIAS <タイプの別名>]
  [EXTENDS <親タイプ/属性名>, ...] ;
【例】
type user_ftp_t extends ftp_anon_t ;
```

図６　TYPE 構文の拡張

(2)の TYPEEXTENDS 構文は、前方参照を行う必要がある場合に **TYPE** 宣言時には親タイプが未定義扱いとなってしまうことを回避するために追加された。

この構文は、定義済みのタイプ同士の親子関係を新たに定義する。ポリシーソースファイルの構成上、子タイプの宣言よりも被継承側である親タイプの宣言が後ろにくる場合もあるため、タイプ宣言順序に依存せずに親子関係を定義するためにこの構文を使用する。

```
TYPEEXTENDS <定義済みタイプ名>
  EXTENDS <タイプ/属性名>, ... ;
【例】
typeextends user_home_t
      extends samba_share_t;
```

図７　TYPEEXTENDS 構文の拡張

(3)の@プレフィクスは、ALLOW 文などでアクセス権限を設定するタイプを指定する際に

対象となるタイプは派生したタイプも含むことを明示的に指定するために使用する。
タイプ/属性名の前に@を付加すると、タイプ/属性の指定は継承された子タイプを全て含むように拡張される。
これにより、子タイプへ継承させる権限を明示的に選択することが可能で、意図しない権限の付与を抑制することができる。
例えば、図８のようなタイプの親子関係が存在するとき、@プレフィクスは構文例のように作用する。

```
<Type A>
  ├<Type B>
  │   └<Type C>
  └<Type D>
      └<Type E>
```

【 構文例 】

```
ALLOW foo_t @A:file read;
⇒ ALLOW foo_t {A B C D E}:file read;
  と等価
ALLOW foo_t @A - @B:file read;
⇒ ALLOW foo_t {A D E}:file read;
  と等価
ALLOW @D self:process signal;
⇒ ALLOW {D E} {D E}:process signal;
  と等価
TYPE_TRANSITION @B @D:process A;
⇒ TYPE_TRANSITION {B C} {D E}
            :process A;    と等価
```

図８　@プレフィクスの作用

ALLOW 文のみならず、タイプ集合を記述することを許容する以下の構文において、@プレフィクスを使用して子タイプも含めた権限の付与を行うことができる。

- Allow 文
- AuditAllow 文/AuditDeny 文/DontAudit 文
- NeverAllow 文
- Type_Transition 文/Type_Change 文/Type_Member 文
- Constrain 条件文
- Role 文/Role_Transition 文

なお、checkpolicy は図 9 に示すセキュリティポリシーの生成処理の中で、make と m4 によって単一の中間ファイルに統合されたポリシーソースをコンパイルし、最終的にカーネルにロード可能なバイナリ形式を出力する。
本節で述べた構文の拡張は、全て checkpolicy が直接解釈する構文に対する拡張である。

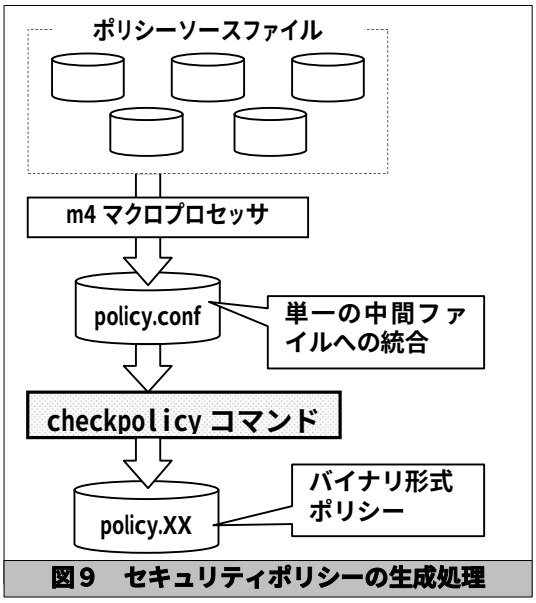

図９　セキュリティポリシーの生成処理

4-2　checkpolicy の追加実装

今回の拡張における実装上の主要な変更点は、タイプ情報を保持するデータ構造の利用方法と、構文から渡されたタイプ名の解釈の２つである。変更行数の合計は全部で 400 行程度に収めることができた。

最初に、タイプ間の親子関係を表現するため、既存の属性(attribute)の実装を拡張した。
checkpolicy の内部では、タイプまたは属性の情報を保持する目的で type_datum 構造体を共通に使用している。

```
typedef struct type_datum {
  uint32_t value;          // タイプ ID
  unsigned char primary; // 別名フラグ
  unsigned char isattr;  // タイプ or 属性
  ebitmap_t types;         // ビットマップ値
} type_datum_t;
```

図 10　type_datum 構造体の定義

type_datum には、属性をアタッチしているタイプの一覧を保持するための ebitmap_t 型のメンバ types が含まれるが、type_datum がタイプの情報を格納している時にはこのメンバは未使用となっている。
今回の拡張おいて、従来は専ら属性のアタッチメントを表現するために使用されていた types を、このタイプを継承している子タイプの一覧を保持するために使用した。

この構造体は checkpolicy とは別の libsepol に由来するものであるが、データ構造の変更を伴わずに当該機能を実装したことで、このライブラリを使用する他のアプリケーションは今回の拡張に伴う影響を受けない。

次に、子タイプも含めたアクセス権の付与を行うために、構文に含まれるタイプ名をスキャンする set_types()関数に拡張を加え、@プレフィクスをタイプ名の前に発見した時は、上記の子タイプ一覧情報を利用して再帰的に権限を付与するように変更を行った。

4-3 従来バージョンとの互換性

checkpolicy コマンドはテキストで記述されたセキュリティポリシーを、カーネルにロード可能なバイナリ形式に変換する。

従来バージョンとの上位互換性には、従来の形式で記述されたセキュリティポリシーを正しく解釈できることと、出力したバイナリを従来のカーネルでも正しくロードできることが期待されているが、本論文で提案した checkpolicy の拡張はこの２つの条件を完全に満たしている。

４-１節で述べた拡張文法を使用していない従来のセキュリティポリシーを、今回の拡張を加えた checkpolicy でコンパイルした場合、記述したポリシーは従来のものと同様に解釈されるため、既存のセキュリティポリシーの資産は影響を受けない。

checkpolicy によって生成されたバイナリの中では、TYPE〜EXTENDS 構文や@プレフィクスを使用して記述された新しいセキュリティポリシーも、親子タイプは互いに共通のセキュリティ属性を持つ複数のタイプであるように見えるので、バイナリレベルでの互換性は完全に保たれている。

このため、カーネルへのポリシーのロードやポリシー分析ツールは既存のものをそのまま使用することができる。

例えば、<Type A>を継承する<Type B>と<Type C>が存在するとき、
「allow X @A:file read;」
をコンパイルして出力されるバイナリは
「allow X {A B C}:file read;」
と見えることとなり、従来のバイナリで十分に表現可能である。

5. タイプ継承を用いたポリシー設定例

図９のように、SELinux のセキュリティポリシーは基本的にはアプリケーション毎に分割して記述され、make 時にこれらを単一のファイルに統合して checkpolicy に渡される。

図 11 は標準セキュリティポリシーのソースツリーのうち主要なファイルの構成であるが、Apache や Samba などの主要なアプリケーション毎のポリシー設定は、domains/program 以下のディレクトリに分割して格納されている。

各アプリケーションが継承されたタイプに対するアクセス権限を持つためには、アクセス権限の付与に際して@プレフィクスを付与してアクセス許可を行わなければならない。

そのためにはアプリケーションの動作に着目して、他のドメインから共通にアクセスされる可能性のあるタイプと、自ドメインからのみアクセスを希望するタイプに分類し、共通にアクセスされるタイプに対するアクセス許可には@プレフィクスを付与することで実現できる。

```
<Policy Tree's Root>
 ├Makefile
 ├assert.te
 ├attrib.te
 ├constraints
 ├domains/
 │   ├ admin.te, user.te
 │   └ program/*
 ├file_contexts/
 │   ├ types.fc
 │   └ program/*
 ├macros/
 │   ├ <標準マクロ群>
 │   └ program/*
 ├rbac
 ├types/
 └users
```

図 11　セキュリティポリシー・ソースツリー

現在 NSA で管理されている標準セキュリティポリシーは、まだタイプ継承を意識したコーディングになっていないが、将来的には各

アプリケーション向けのポリシー設定においても、ユーザ側が定義済みタイプを継承して独自のタイプを定義することを意識したポリシーの記述を行うべきと考える。

本章では、タイプ継承を利用した独自のセキュリティポリシーの設定例として、図 12 に示すディレクトリに対して HTTP/FTP/Samba サーバからアクセスする場合のアクセス許可設定例を示す。

```
/var/                : var_t
└ www/               : httpd_sys_content_t
   └ html/           : httpd_sys_content_t
      ├ images/      : www_images_t
      └ logs/        : www_logs_t

※ www_images_t と www_logs_t に相当するタイプは NSA 標準ポリシーには含まれない。
```

図 12　ポリシー設定ディレクトリ例

図 12 において、/var/www/html は標準的な HTTP サーバのルートドキュメントで、NSA の標準セキュリティポリシーにも HTTP サーバのドメインからアクセスするための設定が含まれている。

加えて、images と logs というディレクトリを新たに追加して以下のセキュリティポリシーを独自に設定する。

- images/以下には HTTP サーバから読み込み専用でアクセスする必要があり、同時に Samba/FTP サーバからは読み書き可能アクセスを行うことができる。
- logs/以下には、CGI スクリプトから追記のみ可能で読み込みを許してはならない。加えて、FTP サーバから読み込み専用でアクセスすることができる。

HTTP サーバの設定例

```
type httpd_sys_content_t,
    file_type, sysadmfile;
type httpd_sys_script_ao_t,
    file_type, sysadmfile;
r_dir_file(httpd_t
          ,@httpd_sys_content_t)
allow httpd_sys_script_t
    @httpd_sys_script_ao_t : file
    {ioctl getattr lock append}
allow httpd_sys_script_t
    @httpd_sys_script_ao_t
    dir : ra_dir_perms ;
```

HTTP サーバの設定には、サーバ自身がコンテンツを読み込む権限と、CGI スクリプトがファイルへ追記する権限が必要である。これらは以下の通りに、継承を意識したアクセス権限の付与が行われているものとする。なお、HTTP サーバは httpd_t ドメインで、CGI スクリプトは httpd_sys_script_t で動作している。

httpd_sys_content_t は Web コンテンツに対して、httpd_sys_script_ao_t は CGI スクリプトの追記専用[5]ファイルに対して付与され、それぞれ継承を意識したアクセス権限の付与を行う。

FTP サーバは images ディレクトリ以下に読み書き可能アクセスを行い、logs ディレクトリ以下に読み込み専用アクセスを行う必要がある。ここでは、FTP サーバの読み込み専用タイプとして ftpd_file_ro_t と、読み書き可能タイプとして ftpd_file_rw_t を定義して、それぞれに継承を意識したアクセス権を付与する。

FTP の対象ディレクトリまでのパスにはディレクトリを探索するための権限が必要であるので、属性として ftpd_file_path を定義してディレクトリを探索するための最小権限を付与する。

FTP サーバの設定例

```
attribute ftpd_file_path;
type ftpd_file_ro_t
    extends ftpd_file_path,
    file_type, sysadmfile;
type ftpd_file_rw_t
    extends ftpd_file_ro_t,
    file_type, sysadmfile;
allow ftpd_t @ftpd_file_path :
              dir {search getattr};
r_dir_file(ftpd_t, @ftpd_file_ro_t)
create_dir_file(ftpd_t,
                @ftpd_file_rw_t)
```

Samba サーバは images ディレクトリ以下に読み書き可能アクセスを行う必要がある。FTP サーバと同様に、読み込み専用タイプと

[5] write 権限に換えて append 権限を付与しているため、既存のファイル内容を改ざんできないポリシーとなる。なお、NSA の標準ポリシにはこのような権限を一括して付与するためのマクロは含まれていない。

して samba_file_ro_t と、読み書き可能タイプとして samba_file_rw_t を定義して、それぞれに継承を意識したアクセス権を付与する。samba_file_path は、Samba 共有ファイルへのアクセスパスである。

Samba サーバの設定例

```
attribute samba_file_path;
type samba_file_ro_t
    extends samba_file_path,
    file_type, sysadmfile;
type samba_file_rw_t
    extends samba_file_ro_t,
    file_type, sysadmfile;
allow smbd_t @samba_file_path :
              dir {search getattr};
r_dir_file(smbd_t,@samba_file_ro_t)
create_dir_file(smbd_t,
                @samba_file_rw_t)
```

各アプリケーション側での以上の設定を利用して、/var/www/html 以下に適切なアクセス権を付与する。

将来的にはこれらの設定は NSA 標準セキュリティポリシーにマージされ、ユーザはこれらのタイプを自由に継承して利用することが期待される。

FTP/Samba には、httpd_sys_content_t ディレクトリを経由して images/logs ディレクトリにアクセスするためのための権限を付与する。images 以下には HTTP サーバの読み込みアクセスと FTP/Samba サーバの読み書き可能アクセス権限を付与し、logs 以下には CGI スクリプトの追記専用アクセス権限と FTP サーバの読み込み権限を付与する。

/var/www/html 以下の設定例

```
# For Access Path
typeattribute httpd_sys_content_t
              ftpd_file_path,
              samba_file_path;
# For /var/www/html/images
type www_images_t
    extends httpd_sys_content_t,
            ftpd_file_rw_t,
            samba_file_rw_t;
# For /var/www/html/logs
type www_logs_t
    extends httpd_sys_script_ra_t,
            ftpd_file_ro_t;
```

以上が、タイプの継承を用いて必要な権限を付与した場合の設定例である。

比較のため、以下にタイプの継承を用いないで同じセキュリティポリシーの設定を行うための記述例を示す。

/var/www/html 以下の設定例（継承を使用しない場合）

```
# For Access Path
allow ftpd_t httpd_sys_content_t
      dir : {search getattr};
allow smbd_t httpd_sys_content_t
      dir : {search getattr};
# For /var/www/html/images
type www_images_t,
      file_type, sysadmfile;
r_dir_file(httpd_t,www_images_t)
create_dir_file(ftpd_t
                , www_images_t)
create_dir_file(smbd_t,
                , www_images_t)
# For /var/www/html/logs
type www_logs_t,
      file_type, sysadmfile;
allow httpd_sys_script_t
    www_logs_t : file
    {ioctl getattr lock append}
allow httpd_sys_script_t
    www_logs_t : dir
    {read getattr lock search ioctl }
r_dir_file(ftpd_t, www_logs_t)
```

以上の通り、従来通りのセキュリティポリシーの記述に比べ、タイプ継承を用いた記述は、複数タイプ間で共通に保持すべきアクセス権限を親タイプ側で一括して付与することで、明らかにセキュリティポリシーの記述量が少なくなっていることがわかる。

6. まとめ

近年のセキュリティ問題に対する関心の高まりに伴い、SELinux に対する期待も高まっているが、セキュリティポリシー記述の困難さが普及を妨げる要因の一つとなっている。

このような問題に対し、継承に基づくタイプ定義には以下のようなメリットがある。

- 複数のタイプが共通に保持しなければならないアクセス権限を、より上位のタイプで一括して付与することができる。これによ

りポリシー設定の冗長性を排除可能で、一貫性と可読性の向上に寄与する。
- 各アプリケーションの専門家によって設定されたポリシーを、ユーザ独自の設定に引き継ぐことができる。これにより、各ユーザに対してアプリケーションの挙動やセキュリティポリシーに対する高度な知識を要求せずに済む。

本稿の成果物は、現在もアップストリームへのマージを目指して SELinux コミュニティでの議論が継続中である。

今後は、ポリシーコンパイラだけでなく、NSA の管理する標準セキュリティポリシーにタイプ継承を意識した記述を組み込み、ポリシーを設定するユーザ側で必要なタイプを組み合わせて容易に目的のタイプを生成できるようにするための基盤作りが課題となる。

7. 参考文献

[1] Linux Kernel Source
http://www.kernel.org/

[2] NSA SELinux
http://www.nsa.gov/selinux/

[3] R.Spencer, S.Smalley, P.Loscocco, M.Hibler, D.Andersen, and J.Lepreau. **The Flask Security Architecture: System Support for Diverse Security Policies.** (1999)
http://www.cs.utah.edu/flux/papers/flask-usenixsec99.pdf

[4] P.Loscocco and S.Smalley, **Integrating Flexible Support for Security Policies into the Linux Operating System.** (2002)
http://www.nsa.gov/selinux/papers/slinux-abs.cfm

[5] S.Smalley, **Configuring the SELinux Policy** (2002,2005)
http://www.nsa.gov/selinux/papers/policy2-abs.cfm

[6] 女部田武史, SELinuxBOF(2004)
「なぜ SELinux の設定は難しいのか」
http://www.selinux.gr.jp/documents/2004bof.html

[7] 中村雄一、上野修一、水上友宏
「SELinux 徹底ガイド」日経 BP(2004)
Part.3 SELinux をより一層理解する