卒業論文　　2005年度 (平成17年度)

システム構成とアプリケーション設定に基づく セキュリティ・ポリシ自動生成フレームワークの構築

慶應義塾大学 環境情報学部

氏名：山本　雄太

指導教員

慶應義塾大学 環境情報学部

村井 純

徳田 英幸

楠本 博之

中村 修

高汐 一紀

湧川 隆次

平成18 年2 月13 日

卒業論文要旨 2005 年度 (平成 17 年度)

アプリケーション設定とシステム構成に基づく セキュリティ・ポリシ自動生成フレームワークの構築

論文要旨

セキュア OS は強制アクセス制御と最小特権の 2 つの機能を用いて、攻撃から資源を保護することができる。また、たとえ攻撃を受けても被害を最小限に留めることが可能である。しかし一方、厳格にアクセス制御を行うために複雑な設定が必要である。そのため、矛盾のない設定が困難である。

セキュア OS の一つである SELinux は、セキュリティ・ポリシと呼ばれるアクセス制御の定義によって、非常に細かい粒度でアクセス制御を行うことができる。しかし、非常に細かな粒度に対する設定が必要なことから、その利用は容易ではない。

本研究では、セキュリティ・ポリシの自動生成を提案する。セキュリティ・ポリシに精通している者があらかじめ定義したセキュリティ・ポリシをベースとして自動調整を行う。調整はアプリケーションの設定ファイルとシステム構成に基づいて行う。これにより、前提となる知識を必要としないセキュリティ・ポリシの自動生成が可能となる。

また、自動生成を実現するためのフレームワークを設計した。セキュリティ・ポリシのベースとなるポリシ・テンプレートと、その自動調整を行う調整スクリプトに関して定義した。実例として、本フレームワークに準じた Apache のセキュリティ・ポリシの自動生成を実装した。

本研究により、システム構成とアプリケーション設定に基づくセキュリティ・ポリシの自動生成を実現した。これにより、セキュリティ・ポリシを定義する労力が削減され、SELinux に精通していない者であっても SELinux を容易に導入することを可能にした。

キーワード
1. セキュア OS, 2. SELinux, 3. セキュリティ・ポリシ

慶應義塾大学 環境情報学部

山本 雄太

Abstract of Bachelor's Thesis

Academic Year 2005

Framework for automated Security Policy builder using application and system configuration

Secure OS protect resources from attacks using its two functions; mandatory access control(MAC) and least privilege. It can minimize the damages of attacks even when attacks succeed. However, in order to manage strict access control, secure OS requires complicated configuration, and it is difficult to configure without any contradiction in the configuration file.

SELinux is one of the secure OSes. SELinux, as in the cases of other secure OSes, also requires security policy, which is access control definition. It can perform a very fine-grained access control compared to other secure OSes. However, it also requires a fine-grained security policy, which makes the configuration very complicated.

In this research, automatic generation of security policies for SELinux is proposed. It automatically adjusts the configuration from a basic security policy, which is defined by knowledgeable person. Adjusting is done based on the application configuration files and system configuration. As a result, automatic generation of a security policy that does not require certain premises is realized.

A framework to realize automatic generation is also designed in this paper. Policy templates, which can work as the base of the security policy and scripts for automatic adjustments are defined. As an example, automatic generation of security policy for Apache httpd is implemented based on this framework.

In this paper, automatic generation of a security policy based on the application configuration files and system configuration is realized. As result, the cost for defining security policy is reduced, which enables nonprofessional person with SELinux install and configure the OS with ease.

Keywords :
1. SecureOS, 2. SELinux, 3. Security Policy

Keio University , Faculty of Environmental Information

Yuta Yamamoto

目 次

図 目 次

表 目 次

第1章　序論

クラッキング、ウィルスやワーム、DDoS 攻撃、SPAM、Phishing 詐欺など、インターネットは様々なセキュリティに関する問題を抱えている。クラッキングなどの悪意を持った攻撃の多くはサービス・アプリケーションの脆弱性を狙うものである。そのような攻撃への対策は重要な課題である。

攻撃を受けた際のホストの被害は、資源への不正なアクセス利用だけではなく、そのホストを踏み台に利用して別のホストに攻撃するものも少なくない。つまり、ホストが攻撃者に乗っ取られることで第二第三の攻撃へと加担する恐れがある。

これらの問題に対処する研究の一つにセキュア OS[1] がある。セキュア OS は従来の OS には存在しなかった強制アクセス制御と最小特権を実現する。

強制アクセス制御は従来の OS のアクセス制御と異なる。粒度の細かい制御を提供するこの制御の特徴は、あらかじめ管理者が定義したセキュリティ・ポリシに基づいて厳格にアクセス制御が行われる点である。すなわち、ユーザ自身が作ったリソースであっても自由にアクセスできるわけではない。したがって、管理者であってもセキュリティ・ポリシに反するアクセスは許可されない。

最小特権は、ユーザやプロセスに対して必要最低限の権限のみを与える機能である。従来の OS は、特権ユーザに全ての権限が集中している。そのため、特権ユーザの権限が奪われてしまうと攻撃者に全てのアクセスを許可してしまう問題がある。しかし、システム管理ユーザ、ログ管理ユーザのように特権の分割管理を行えば、この問題は解決できる。これが最小特権である。正しく権限を分割すれば、たとえ権限が奪われても、奪われた権限が許す範囲でしか影響を及ぼすことがない。

セキュア OS はサービス・アプリケーションの脆弱性に対する攻撃に対抗することができる。セキュア OS はサービス・アプリケーションに本来必要な権限しか与えない。そのため、攻撃を受けたとしてもホストの管理者権限が乗っ取られてしまう可能性は低い。このことは、脆弱性を狙った攻撃に対する強力な対策となる。また、シェルの実行権限のような攻撃を成功させるために必要な権限が与えられておらず、攻撃を未然に防ぐことができる場合も多い。セキュア OS は昨今問題になっている未知の脆弱性を狙う 0-day 攻撃の有効な対策になる。このようにセキュア OS は高い安全性を実現する。

一方でセキュア OS は、通常の OS に必要な設定とは別にアクセス制御を行うための定義を必要とする。本研究では、このアクセス制御の定義をセキュリティ・ポリシと呼ぶ。セキュア OS が実現する高い安全性はこの定義に基づいている。そのため、正しく定義されなければ攻撃を許してしまう場合がある。また、システム全体が正常に動作しなうなる場合もある。このように、アクセス制御の定義はシステムに対する強い影

響力がある。そのため、アクセス制御の粒度が細かくなればなるほどその定義は複雑になっていく。

今日では様々なセキュア OS が開発されている。National Security Agency（以後、NSA と呼ぶ）の開発している Security-Enhanced Linux（以後、SELinux と呼ぶ）[2] や、LIDS Project の開発している LIDS[3]、Trusted BSD Project の開発している Trusted BSD[4] などのオープンソースのセキュア OS から、商業ベースで開発されている Sun の Trusted Solaris[5]、Infocom の Pitbull[6] などがある。

SELinux はオープンソースのセキュア OS である。SELinux は非常に細かな粒度でのアクセス制御が可能である。また、柔軟性に富んだセキュリティ・アーキテクチャを採用している。そのため、様々な環境や条件に対して適切なアクセス制御を行うことが出来る。しかしその一方で、粒度が細かくまた柔軟性に富んだアクセス制御を実現するために、必要なアクセス制御の定義が非常に複雑である。

1.1 本研究の目的

セキュア OS は様々な攻撃に対処しうる高い安全性を持つ。しかし、同時にセキュリティ・ポリシの定義にあたって高いスキルと労力を必要とする。そのため、ごく限られた環境で用いられることはあっても、一般的なサーバに用いられることはないのが実情である。SELinux においても同様に、セキュリティ・ポリシの定義は重要な課題とされてきた。

そこで本研究では、システム構成とアプリケーションの設定に基づくセキュリティ・ポリシの自動生成を提案する。管理者がセキュリティ・ポリシを定義するコストを削減する。

また、セキュア OS は定まった目的のために構築されたサーバでは比較的導入しやすい。だが、学習・実験サーバのような多様な目的で運用されているサーバにおいては、日々サービスやソフトウェアに求められる動作が変化する。そのため、セキュリティ・ポリシの変更が頻繁に行われ、SELinux の運用コストが大きくなりすぎるため、なかなか導入されない。本研究は、自動生成によってその時に求められているセキュリティ・ポリシを再定義することを容易にする。

1.2 本論文の構成

本論文では、第 2 章において SELinux の概要を述べ、SELinux に関する要素技術と関連研究を整理する。第 3 章において、SELinux の問題点を整理する。第 4 章において、セキュリティ・ポリシの定義の問題点を解決する自動生成のアプローチについて議論する。また、フレームワークのモデルの提案と設計を行う。第 5 章で具体的な実装方法について述べる。第 6 章では本システムの評価を行う。自動生成されたセキュリティ・ポリシの正当性とセキュリティ強度を評価する。また、自動生成を行うための工数についても評価を行う。第 7 章で本研究の成果と今後の課題を挙げる。

第2章　SELinux

SELinuxは、非常に細かい粒度でアクセス制御を行うことができるオープンソースのセキュアOSである。本章では、SELinuxの概要を述べ、基礎技術を整理した上で、そのセキュリティ・ポリシについても述べる。

2.1　SELinuxの概要

SELinuxはNSAによって開発されたセキュアOSで、2000年にGPLのもと公開された。Flask[7]と呼ばれるセキュリティ・アーキテクチャをベースとする。Flaskは柔軟性の高いアクセス制御を行うために必要なアーキテクチャである。Flaskに必要な柔軟性としてアクセス権限の継承をコントロールすること、粒度の高いアクセス制御を行うこと、一度定めたアクセス権限を取り消すことが出来ることなどが挙げられている。FlaskはNSAによってフレキシブルなアクセス制御を行うために開発が進められてきた。また、現在ではLinux2.6カーネルから標準導入されたセキュリティ機能拡張インタフェースであるLinux Security Module（以後、LSMと呼ぶ）を用いて実装されている。そのため、Linux環境に大きな変更を加えることなく導入することができる。

SELinuxは、Type Enforcement（以後、TEと呼ぶ）とRole Based Access Control（以後、RBACと呼ぶ）の2つのアクセス制御機能を持ち、非常に粒度の細かいアクセス制御を可能としたセキュアOSである。TEはプロセス・リソース間の最小特権を実現する。同様に、RBACはユーザ・リソース間の最小特権を実現する機能である。最小特権とは、ユーザやプロセスなどに必要最小限の権限のみが与えられることである。TEの詳細については第2.2.2節で、RBACの詳細については第2.2.3節で述べる。SELinuxが行うアクセス制御の概要を図2.1に示す。SELinuxはLinuxの行う従来のパーミッションチェックに加えて、独自のアクセス制御を行う。このアクセス制御は、リソースに付与されたSecurity Contextと呼ばれるラベルを用いて行われる。図2.1における属性がラベルである。アクセス制御は、あるラベルからあるラベルへのアクセスを許可・制限することで定義される。

また、SELinuxは現在も活発に開発が進められている。新たに、Mult-Level Security（以後、MLSと呼ぶ）やMulti-Category Security（以後、MCSと呼ぶ）などの追加が計画されている。MLSとは、オブジェクト（リソース）を極秘、秘、公開など、機密レベルによって分類し、サブジェクト（プロセス）に与えられたクリアランス（権限）を元にアクセス制御を行う概念である。MCSは、MLSを単純化しオブジェクトをカテゴリによって分類してアクセス制御を行う。

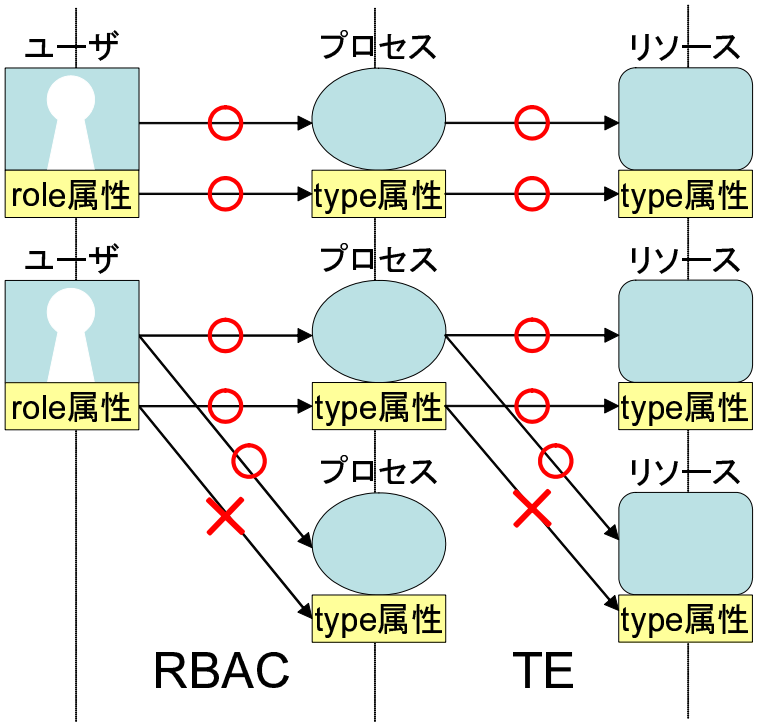

図 2.1: 強制アクセス制御の概要図

2.2 セキュリティ・ポリシ

セキュリティ・ポリシは、SELinux が行うべきアクセス制御を定義するものである。大別すると、後述する Security Context、TE、RBAC の定義から構成される。

また、セキュリティ・ポリシは実際に利用されるときにはポリシ・バイナリにコンパイルされて利用される。ただし、Security Context の定義の一部はサービス・アプリケーションで分割されたファイルを一つにしたポリシ・ソースの状態のままで利用される。

現在、セキュリティ・ポリシを制御を行う方針によって分類すると strict と targeted と呼ばれる 2 つのタイプに分けられる。strict は TE と RBAC の両方を活用して厳格にアクセス制御を行うポリシである。規定されていないアクセスは拒否される。そのため、許可したいアクセスは全てセキュリティ・ポリシとして記載する必要がある。一方、targeted は規定されていないアクセスは許可され、セキュリティ・ポリシが定義されているサービス、アプリケーションでのみ強制アクセス制御を行う。SELinux の導入コストを下げるために、比較的緩やかにアクセス制御を行うポリシである。また、targeted では RBAC が採用されていない。ユーザに与える権限を SELinux で制御した場合、ユーザは SELinux に対する知識を必要とする。これを避けるためである。

基本的なサービスのセキュリティ・ポリシは標準で SELinux に添付されている。しかし、粒度の細かなアクセス制御を行う SELinux の性質上、添付されているセキュリティ・ポリシは全ての環境で利用できるものではない。ディストリビューション毎の違

いだけでも、採用している SELinux のバージョンの差異、ファイルパスの差異、サービスの実装の差異がある。そのために、セキュリティ・ポリシの定義が異なる。現在では、添付される標準のポリシは各ディストリビューション毎やプロジェクト毎に変更・管理されているのが普通である。

以下に、セキュリティ・ポリシを構成する Security Context、TE、RBAC を説明する。

2.2.1 Security Context

SELinux はプロセスなどの OS の管理するリソースにそれぞれラベルを付与し、ラベルを単位としたアクセス制御を行う。このリソースを抽象化したラベルを Security Context と呼ぶ。SELinux はこの Security Context を用いたアクセス制御で、セキュア OS の 2 つの要素である強制アクセス制御と最小特権を実現する。Security Context を定義することで、抽象化されたラベルに具体的な値を設定することができる。

Security Context は図 2.2 のように記載される。左から、user 属性、role 属性、type 属性となる。全てのリソースにはこれらの属性が定義される。

```
root:sysadm_r:sysadm_t
```

図 2.2: Security Context

user 属性とはユーザを識別するための属性である。しかし、この user 属性は従来の Unix のユーザと必ずしも同じものではない。例では root とされているが、一般に、明示的に定義されなかったユーザは、デーモンなどであれば system_u、一般ユーザであれば user_u といった属性にまとめられ定義される。後述する role 属性は、この user 属性を単位としてまとめられる。user 属性は SELinux のセキュリティ・ポリシ上で定義される。

role 属性は後述する RBAC に関係した属性である。role 属性はシステム管理やログ管理など行う業務毎に定義される。ユーザには常に何か一つの role 属性が付与される。ユーザは自身に付与されている user 属性に許された role 属性の中でしか role を切り替えることができない。

type 属性は TE に関係した属性である。TE はアクセス制御の最小単位であるプロセス・リソース間のアクセス制御を行う。そのため、type 属性はラベルの最小単位であるといえる。特に、プロセスに対して付与される type 属性を domain、それ以外のリソースに対して付与される type 属性を type と呼ぶ。これらは role 属性を単位としてまとめられる。domain の場合はそれぞれ対応する role にまとめらる。type の場合は特に object_r ロールにまとめられる。

実際のシステムにおけるそれぞれの関係を、図 2.3 に示す。

Security Context を付与する一例として file_contexts の定義を説明する。file_contexts の定義は図 2.4 のように正規表現を含んだ実際の path と付与する Security Context に

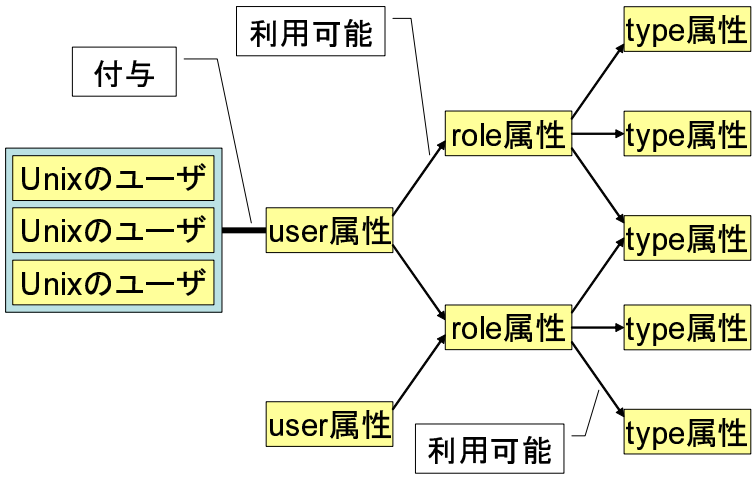

図 2.3: Security Context の属性間の関係

よって定義される。path と Security Context の間に file の種別を指定するためのオプションが含まれることもある。--ならば通常のファイル、-d でディレクトリといったものである。図 2.4 であれば、/usr/sbin/httpd は Apache の実行バイナリで通常のファイルであるため"--"がオプションとして設定されている。

```
/var/www(/.*)?          system_u:object_r:httpd_sys_content_t
/usr/sbin/httpd    --   system_u:object_r:httpd_exec_t
```

図 2.4: Security Context の定義

Security Context は、ファイル以外にもファイルシステム自体や、ネットワーク関係ではポート、ネットワークインタフェースなど OS の管理する様々なリソースにあわせて定義の方法が存在する。例えば、ファイルであれば file_contexts、ネットワークリソースであれば net_contexts、ファイルシステム自体であれば fs_use などがある。また、変則的なものとしては proc ファイルシステムのように、恒久的にマップ情報が利用できず、また作成元プロセスよる付与や type 遷移も行えないものは、genfs_contexts で定義されている。

また、Security Context はあらかじめ定義を行うだけではない。chcon コマンドを用いて、セキュリティ・ポリシが許す範囲内で Security Context の再設定が可能である。これは権限の範囲内で管理者ではないユーザにも可能である。

2.2.2 Type Enforcement

TE はサブジェクト（プロセス）からオブジェクト（リソース）へのアクセス制御を行う機能である。第 2.2.1 節で述べた Security Contexts の type 属性を用いる。SELinux

は TE により各プロセスを必要最低限の権限で動作させることを可能にする。すなわち、プロセスにおける最小特権の実現である。TE の概要を図 2.5 に示す。

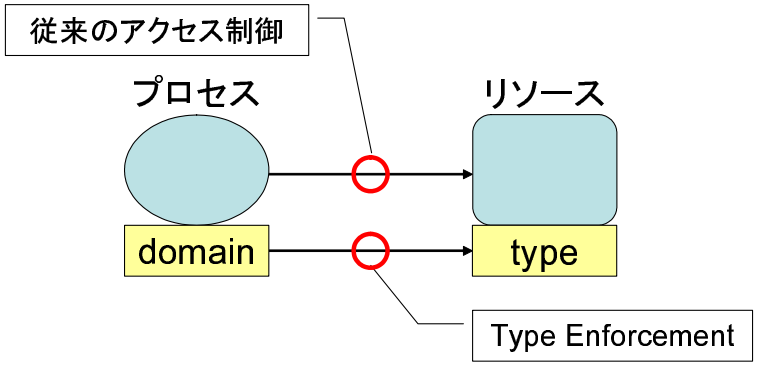

図 2.5: Type Enforcement の概要図

TE に関わる SELinux の重要な要素に domain 遷移がある。SELinux は、ある domain からある domain への遷移を特定の type のファイルが実行されることをきっかけにして domain を遷移させる。例えば、httpd がサービスとして起動される際の domain 遷移の場合は、initrc_t ドメインから httpd_exec_t タイプのファイルをエントリポイントとして、httpd_t ドメインへと遷移する。domain 遷移の概要を図 2.6 に示す。

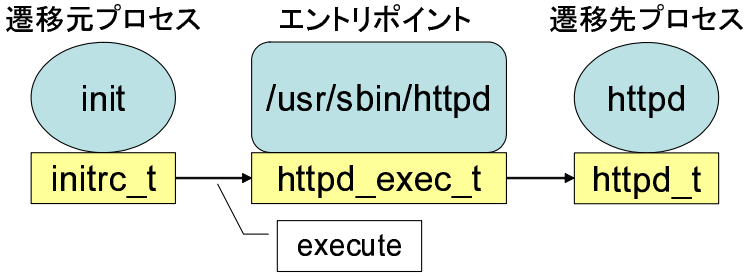

図 2.6: domain 遷移の概要

各リソースは file, lnk_file, dir, process, socket などのようなオブジェクトクラスに分けられ、各オブジェクトクラス毎に read, write, append などの permission が設定されている。SELinux では、アクセス制御を行うために domain ごとに各 type に対して許可される permission を定義する。これをアクセスベクタと呼ぶ。TE は定義されたアクセスベクタに従ってアクセス制御を行う。

例として、user_t ドメインが付与されている一般ユーザのシェルを取り上げる。domain 遷移の定義が行われているプログラム以外は親プロセスと同じ user_t で実行される。つまり、一般ユーザのシェルから実行されたコマンドのプロセスには user_t ドメインが付与される。cat コマンドであれば、user_t ドメインの持つ権限の範囲内で、ファイルを読み込むことが許可される。勿論、thunderbird[8] のなどのように用途が限定されて

いるものであれば、domain 遷移を行うべきである。なぜなら、一般ユーザのシェルに付与された権限は最小特権という観点から見れば比較的大きな権限だからである。

2002 年に流行した Slapper[9] ワームの場合、Apache[10] の OpenSSL[11] のバッファ・オーバー・フロー脆弱性を突くことでシェルを実行し攻撃コードを実行していた。SELinux を導入し、domain 毎に適切なアクセス権限を定義することで、この攻撃は防ぐことができる。OpenSSL の脆弱性を突かれても、Apache に付与されている httpd_t ドメインはシェルを実行する権限を持っていないからである。そのため、攻撃コードを実行することはできない。また、たとえシェルを実行する権限が与えられていたとしても、被害は Apache の権限の範囲内に収まる。Apache 用のポート以外を Listen することはできないし、もちろんシステムの重要なディレクトリに対して変更を加えることもできない。このように、TE によるプロセスの最小特権を実現することで、攻撃に対処することができる。

TE に関わるセキュリティ・ポリシの定義について以下に取り上げる。

type の定義

type の定義はアクセスベクタを記述する上で必要不可欠である。TE はリソースに付与されたラベルを用いてアクセス制御を行う。そのため、まず最初にラベル付けに用いる type について定義しなければならない。その際、domain と type は特に区別されない。しかし、domain はサブジェクト（プロセス）に付与されるサブジェクト（プロセス）に必要な各種の権限を持った type である。そのため、第 2.2.4 節で述べる Attribute に予め定義されている domain 属性を付与する必要がある。

type の定義を図 2.7 に示す。type、定義する type 名、属性と続けて定義する。run_init_t ドメインは SELinux のユーティリティの一つである run_init コマンド用の domain である。前述したように domain 属性が付与されている。また、run_init_exec_t タイプは run_init コマンドの実行ファイルに対して付与される。file_type で通常のファイルであることを示す。同様に exec_type で実行されるファイルであることを示している。また、sysadmfile によって、管理者の role すなわち sysadm_r ロールから完全にアクセスされるべきファイルであることを示している。

```
type run_init_t, domain;
type run_init_exec_t, file_type, sysadmfile, exec_type;
```

図 2.7: type の定義

遷移の定義

TE の重要な要素に domain 及び type の遷移がある。domain の遷移はプロセスに domain を付与するために必要な概念である。

プロセスの Security Context は domain 遷移によってのみ付与される。Linux では、起動時にカーネルから init プロセスと呼ばれるプロセスが開始され、あらゆるプロセスの先祖となる。同様に、SELinux ではカーネルから init プロセスが作られるとき、カーネルの kernel_t ドメインから init_t ドメインへと遷移する。このように、プロセスが作られていく過程で domain を次々に遷移させて必要な domain へと遷移していく。SELinux は domain に対してアクセス権限を設定するため、domain を遷移させることでアクセス権限の変更が行われるのである。この遷移のルールを定義するのが domain 遷移の定義である。図 2.8 は kernel_t ドメインが init_exec_t タイプをエントリポイントとして init_t ドメインへ遷移するルールである。

```
type_transition kernel_t init_exec_t:process init_t;
```

図 2.8: domain 遷移の定義

また、実体のない抽象的なオブジェクトや、アプリケーションによって作られるテンポラリファイルのように実体はあるが常に存在するとは限らないオブジェクトに対して type を付与するのが type 遷移である。図 2.9 の例は、httpd_t ドメインで動作している Apache が tmp_t タイプのファイルとディレクトリを生成する場合の定義である。この場合には、自動的にそのオブジェクトには tmp_t ではなく、httpd_tmp_t タイプが付与される。

```
type_transition httpd_t tmp_t:{ file dir } httpd_tmp_t;
```

図 2.9: type 遷移の定義

type 変更ルールの定義

type 遷移は新たに作成されたオブジェクトに対して type を付与するものである。一方、type 変更ルールは既に存在するオブジェクト（リソース）に付与された type を、ある domain が付与されたサブジェクト（プロセス）が変更するためのルール定義である。このルールの定義は SELinux 向けに作られた、あるいは改良されたプログラムとの組み合わせで初めて意味を持つものである。表記を図 2.10 に示す。

```
type_change cupsd_t system_dbusd_t:dbus cupsd_dbusd_system_t;
```

図 2.10: type 変更ルールの定義

アクセスベクタの定義

ある domain がある type に対して許可されるアクセス（permission）を定義するのがアクセスベクタである。表記を図 2.11 に示す。どのような種別のアクセスベクタであるかが最初に記述される。その後、アクセス元 domain、アクセス先 type、type のオブジェクトクラス、permission と続いて記述する。

```
allow httpd_t etc_t:file { read getattr ioctl };
auditallow sysadm_t security_t:security setenforce;
dontaudit httpd_t self:capability sys_tty_config;
```

図 2.11: アクセスベクタの定義

SELinux では明示的に許可されていないアクセスはシステムによって拒否される。すなわち、全ての許可されるべきアクセスはアクセスベクタによって定義しなければならない。

targeted ポリシでは規定されていないアクセスは許容されると述べた。これは、unconfined_t タイプというポリシの定義されていないリソースに対する type があり、この type を付与されているサブジェクト、オブジェクトは全てのアクセスが許可されるようにアクセスベクタが定義されているからである。

勿論、targeted でポリシが定義されたサービス・アプリケーションは、この unconfined_t タイプに対してアクセス権限を持つようにアクセスベクタが定義されていなければ、unconfined_t タイプのリソースに対してアクセスを行うことはできない。

SELinux では、アクセスベクタが定義されていないアクセスが発生した場合、情報をログとして出力する。SELinux の version にも関わっているが、現在は Linux Audit System を用いてログが保存されるようになっている。これは、第 2.3.1 節で述べる Audit2allow などに利用される。

アクセスベクタの種別は、allow, auditallow, dontaudit の三種類である。allow はアクセスを許可するアクセスベクタである。図 2.11 では、httpd_t ドメインが etc_t タイプの付与されたファイルに対して read, gettattr, ioctl を行うことを許可している。また、アクセスベクタによって許可されたアクセスであってもログに残したいアクセスが存在する場合には、allow の代わりに auditallow を用いる。auditallow を用いると、許可されたアクセスであってもログに出力して保存することができる。図 2.11 では、SELinux のユーティリティの一つである setenforce コマンドを実行した際にログに出力して保存するように定義している。setenforce コマンドは SELinux によるアクセス制御を行うかどうかを変更するコマンドである。そのため、setenforce コマンドの実行をログに残すことは重要である。システムによって拒否されたがログに残したくない場合は dontaudit を用いる。dontaudit を用いるとそのアクセスを拒否し、またログにも残さないことが可能である。

アクセスベクタアサーションの定義

アクセスベクタアサーションは、明示的に許してはいけないアクセスを定義する。アクセスベクタアサーションが定義されていれば、同様の内容のアクセスベクタが万が一追加されても、セキュリティ・ポリシをコンパイルする際にアサーションエラーを出力する。誤ったアクセスベクタを追加するのを防ぐ効果がある。アクセスベクタアサーションは以下のようにアクセスベクタにおける allow を neverallow にすることで定義される。

```
neverallow domain file_type:process *;
```

図 2.12: アクセスベクタアサーションの定義

2.2.3 Role Based Access Control

RBAC はユーザとリソースのアクセス制御を行う機能である。特権ユーザを含めた全てのユーザに第 2.2.1 節で述べた Security Contexts の role 属性を用いて役割を設定することで、リソースに対するアクセスを制御する。従来の Linux では特権ユーザはシステムに対して全てのアクセスを許可されている。RBAC では、webadm_r, logadm_r など必要な作業・権限に応じて役割の分割を行う。これにより、万が一特権ユーザの権限が奪われるようなことがあっても適切なアクセス制御が可能である。具体的には role は第 2.2.2 節で述べた domain を束ねて定義され、ユーザ毎に遷移することのできる role が定められる。全てのユーザはいずれかの role に属することになる。TE で定義されたサブジェクト、オブジェクトのアクセス制御に加え、role によるアクセス制御を追加する。これにより、たとえ TE によって許容されたアクセスであっても、RBAC で許容されない場合、アクセスを拒否することができる。

図 2.13 に RBAC の概要図を示す。図の例では、user 属性である root に、システム管理用の権限を持った sysadm_r ロール、Web 管理用の権限を持った webadm_r ロールなどが定義されている。さらに、webadm_r ロールは Web 管理を行うのに必要な TE の権限を持った幾つかの domain を束ねて定義される。

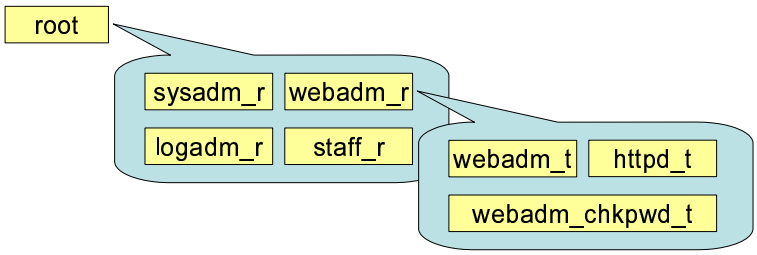

図 2.13: RBAC の概要図

RBAC により、SELinux は各ユーザに対して必要最低限の権限を割り当てることができる。すなわち、ユーザにおける最小特権の実現である。

以下に、具体的なセキュリティ・ポリシにおける RBAC の定義を述べる。

role の定義

RBAC においてユーザに割り当てるための role を定義する。表記を図 2.14 に示す。

```
role sysadm_r types ping_t;
role sysadm_r types passwd_t;
```

図 2.14: role の定義

ある role 属性に対して、割り当てる domain を定義する。この定義は何度定義されても上書きされることがない。上記の例で言えば、sysadm_r ロールは、ping_t ドメインにも passwd_t ドメインにも遷移することができる。role の定義は type とは違い、その定義と同時に割り当てる domain の定義も行う。role の宣言は許可される domain を定義するため、ある domain からある domain へ domain 遷移ルールで domain の遷移が許されていたとしても、遷移後の domain に対して遷移を許可されていない role の場合、domain の遷移を行うことはできない。

遷移許可の定義

role 属性は domain や type と同様に遷移を行う。遷移許可の定義は、ある role がある role に対して遷移するのを許可するルールである。図 2.15 の表記の場合、sysadm_r ロールから system_r ロールへの遷移を許可するものである。

```
allow sysadm_r system_r;
```

図 2.15: 遷移許可の定義

遷移ルールの定義

role の遷移ルールは、新たに作成されるプロセスに対して role を付与するものである。明示的に指定されない場合は、新たに開始されたプロセスには、開始元プロセスに付与されている role がそのまま付与される。ただし、開始されたプロセスに対してアクセス権限を自動的に変更する方法として、role はそのままで、domain のみを遷移させるのが適切である。そのため、このルールは将来的にサポートから外されることになっている。

ユーザの定義

第 2.2.1 節で述べたように、SELinux では Unix のシステムにおけるユーザとは別に user 属性を管理している。ユーザの定義は、この user 属性と遷移を許可する role を定義するものである。表記を図 2.16 に示す。

```
user user_u roles { user_r }
user root roles { user_r sysadm_r system_r };
```

図 2.16: ユーザの定義

user に続いて、システムのユーザアカウントとして登録されているユーザ名を記述する。この際、明示的に登録されなかったユーザは user_u として扱われ、user_u と同じ権限を与えられる。この時、遷移を許可された role に対しては newrole コマンドを用いて role の変更を行うことができる。

2.2.4 その他の定義

以下に SELinux のセキュリティ・ポリシにおけるその他の定義について取り上げる。

Attribute

Attribute は第 2.2.2 節で述べた type の定義に対して付加的に用いる。type に Attribute を関連付ける。type は関連付けられた Attribute と同様の性質を持つ。同じ性質を持ったグループを定義する際に用いられる。Attribute は図 2.17 のように定義を行う。

```
attribute privlog;
```

図 2.17: 属性の定義

Attribute は type と同様に扱うことができ、type と同様にアクセスベクタを定義することで性質を定義する。関連付けられた type は Attribute の性質を付与されるため、汎用的なアクセスベクタを type に関連付けること、type のグルーピングをすることに用いられる。

上記で例として挙げた privlog 属性は標準のポリシの中で定義されている Attribute である。これは syslogd サービスと通信を行う権限を持つ性質の Attribute である。type 定義に付加的に用いることで、その domain に対して syslogd サービスとの通信を許可する権限を追加する。

Constraint

Constraint は TE と RBAC によるアクセス制御に加えて、更に粒度の細かなアクセス制御を行う。TE ではサブジェクト（プロセス）とオブジェクト（リソース）の関係をそれぞれ domain と type の関係で定義するが、constraint は、Security Context を構成する属性の内、type 属性だけでなく、user 属性や role 属性をも含めてアクセス制御を定義することができる。

Conditional

Conditional はポリシを再コンパイルすることなく動作を変更することができる SELinux の機構である。あらかじめ bool 変数をセキュリティ・ポリシで定義しておき、if/else を用いたポリシの適用の選択を行う。get/setsebool を用いてアクセス制御の挙動を変更する。具体的な bool 変数の例を挙げれば httpd_enable_cgi がある。これは apache における cgi の利用の可否に関わる bool 変数であり、この bool 変数を変更することでポリシに変更を加えることなく、アクセス制御を動的に変更することができる。Conditional は SELinux Policy Conditional Extention により実現される。詳細は第 2.3.6 説で述べる。

マクロ

SELinux のセキュリティ・ポリシの定義には GNU m4 マクロプロセッサ [12] によるマクロがサポートされる。マクロを用いることで、セキュリティ・ポリシにおける同様の記述をまとめ、セキュリティ・ポリシが簡潔になる。また、セキュリティ・ポリシの可読性を高める利点がある。

現在の SELinux に標準で添付されているセキュリティ・ポリシには多くのマクロが定義されており、これらを用いることでセキュリティ・ポリシの定義が容易になる反面、マクロの種類とその定義内容を十分に理解しておく必要がある。また、このマクロは独自に定義することも可能である。

2.2.5 具体例

セキュリティ・ポリシの具体例を示す。以下は Fedora Core Linux4[13] の targeted ポリシに含まれる hostname 用のセキュリティ・ポリシである。第 2.2.2 節や第 2.2.3 節で述べた定義の他に、マクロを展開する際に利用される m4 の構文である ifdef が含まれているが、図 2.18 は前述したフォーマットに基づいて定義されている。

hostname はホスト名を表示・設定するための基本的なプログラムの一つである。様々なサービス・アプリケーションから利用され、またユーザも日常的に利用するコマンドであるが、そのセキュリティ・ポリシはこのように複雑なものになっている。

まず daemon_core_rules マクロでデーモンに必要な core ルールを付与した後 (1)、hostname プログラムに必要な様々な権限を付与している。sys_sethostname システムコールを利用するために必要な capability である SYS_ADMIN を許可したり (2)、他のプロセスと通信するために unix_stream_socket を許可したりしている (4)。

```
   % file_contexts/program/hostname.fc
   /bin/hostname  --  system_u:object_r:hostname_exec_t

   % domains/program/hostname.te
(1)daemon_core_rules(hostname, , nosysadm)
(2)allow hostname_t self:capability sys_admin;
(3)allow hostname_t etc_t:file { getattr read };

   allow hostname_t user_tty_type:chr_file rw_file_perms;
   allow hostname_t admin_tty_type:chr_file rw_file_perms;
   read_locale(hostname_t)
   can_resolve(hostname_t)
   allow hostname_t userdomain:fd use;
   dontaudit hostname_t kernel_t:fd use;
   allow hostname_t net_conf_t:file { getattr read };
(4)allow hostname_t self:unix_stream_socket create_stream_socket_perms;
   dontaudit hostname_t var_t:dir search;
   allow hostname_t fs_t:filesystem getattr;

   # for when /usr is not mounted
   dontaudit hostname_t file_t:dir search;

   ifdef(‘distro_redhat’, ‘
   allow hostname_t tmpfs_t:chr_file rw_file_perms;
   ’)
   can_access_pty(hostname_t, initrc)
   allow hostname_t initrc_t:fd use;
```

図 2.18: hostname のセキュリティ・ポリシ

アクセスベクタは図 2.18 のように type を用いて設定されるため、直感的にセキュリティ・ポリシを理解することができない。また、あるファイルを開くために必要な定義はそのファイルが存在するディレクトリにいたる全てのパスにおける search permission や、そのファイル自体への read permission、ファイルディスクリプタに対する use permission が必要となる。図 2.18 の場合は etc_t、すなわち/etc 以下のファイルに対して getattr, read permission を許可している (3)。

しかし、図 2.18 の中には、/etc ディレクトリへ search permission がない。実は daemon_core_rules() のマクロに含まれている。マクロは複数のアクセスベクタを統一的に管理することができる。中でどのようなアクセスベクタが定義されているかは展開された後のセキュリティ・ポリシか、マクロの定義とドキュメントを確認する必要がある。しかし、マクロの定義もマクロの中にマクロが定義されるなど複雑に入り組んでいる。そのため、あるマクロがどのようなアクセスベクタを持っているかを判断するのは難しい。

このように、hostnameのような比較的小さなプログラムであっても、一目ではセキュリティ・ポリシを把握することは難しい。SELinuxやサービス・アプリケーションに精通していない管理者が個別のプログラムに対して正しいセキュリティ・ポリシを定義するのは困難で、現実的に不可能である。

2.2.6 セキュリティ・ポリシの構造

セキュリティ・ポリシは前述したTEやRBACに関する様々な定義の集合体である。実際にアクセス制御に用いられる際には一つにまとめてからコンパイルを行うが、メンテナンス性の向上を目的としてセキュリティ・ポリシは分割して定義される。現在のセキュリティ・ポリシは、以下のような構造を持って定義される。通常、/etc/selinux/(strict|targeted|.*)/src/policy以下に置かれる。図2.19の例はstrictのものである。

新しいサービス・アプリケーションのセキュリティ・ポリシを定義する場合、fcファイルとteファイルが必要である。file_contexts/program以下に.fcファイルを、domain/program以下に.teファイルを作成する。それぞれ、第2.2.2節で述べた方法で、file_contextsとTEの定義を行う。セキュリティ・ポリシをバイナリにコンパイルする際に、.teファイルはpolicy.confに、.fcファイルはfile_contextsに自動的にまとめられて処理される。また、自身でmacroを作成する場合にはmacro/program以下に.teファイルを作成し定義してもよい。定義されたマクロはバイナリにコンパイルする際に自動的に展開される。

個々のサービス・アプリケーションのセキュリティ・ポリシ定義ファイルはこのように分割して定義・管理される。しかし、実際には他のアプリケーションとの依存性の解決に対する明確な指針は設定されていない。

そのため、第2.2.5節で紹介したhostnameのような他のサービス・アプリケーションから利用されるサービス・アプリケーションの定義に問題がある。このような場合、適切にアクセス権限を変更するために、domainの遷移を行う必要がある。これにより、あるdomainからhostname_tドメインに対して遷移の定義を行うためには、あらかじめhostname_tドメインが定義されていなければならないなどの依存関係ができてしまう。

このように、セキュリティ・ポリシはそれぞれのセキュリティ・ポリシ定義ファイルが依存関係にある。そのため、依存関係の解決に注意を払う必要がある。

2.3 関連研究

本節ではSELinuxに於けるセキュリティ・ポリシ作成のコスト低下、ひいてはSELinux導入のコストの削減を目的とした関連研究を取り上げる。

```
% Security Context に関する定義 (.fc ファイルなど)
file_contexts/
file_contexts/program/
file_contexts/misc/
initial_sid_contexts
genfs_contexts
net_contexts
fs_use

% TE に関する定義 (.te ファイル)
domains/
domains/program/
domaine/misc/
macros/
macros/program/
types/
assert.te
attrib.te

% RBAC に関する定義
users
rbac

% 個々のアクセス制御に関する定義
constratints
mls

% 通常変更の必要がないもの
flask/
tunables/
appconfig/
```

図 2.19: セキュリティ・ポリシの構造

2.3.1 Audit2allow

audit2allow は SELinux に標準で含まれるセキュリティ・ポリシ自動生成ユーティリティである。SELinux のアクセス拒否のログから必要なアクセスベクタを自動生成することができる。最も古くから用いられてきたユーティリティであり、現在でも利用

されている。

audit2allow は与えられた SELinux のアクセス拒否のログから、そのアクセス拒否を全て許可する内容のアクセスベクタを自動生成する。そのため、実際にはそのサービス・アプリケーションの動作には必要のないアクセスも含めてアクセスベクタを生成してしまう可能性がある。また、アクセス拒否のログを取った際に行われなかったアクセスに関しては全く考慮されない。そのため、サービス・アプリケーションの正常なアクセスを許可するアクセスベクタが作られない場合がある。このように、audit2allow を用いてセキュリティ・ポリシを生成する場合には、本来不必要なアクセスベクタや必要であるが生成されなかったアクセスベクタを見分けることが必要である。

audit2allow は自動生成ユーティリティであるが、実際の利用にはセキュリティ・ポリシに関する知識を必要とする。

2.3.2 SEFramework

SEFramework[14] はセキュリティ・ポリシ開発用フレームワーク及び、統合環境である。作成するのも解析するのも複雑で難しいセキュリティ・ポリシを、アプリケーションの開発者にとって扱いやすくすることを目的としている。

実現方法として、SEFramework Language と SEFramework tools が提案されている。SEFramework Language はアクセス制御に必要な情報を定義するための抽象化された言語である。同様に、SEFramework tools はモデル図を用いてアクセス制御に必要な情報を定義するものである。しかし、これらは今も開発中であり、詳細な情報は公開されていない。

2.3.3 Automate Policy Generation

Automate Policy Generation（以後、polgen と呼ぶ）[15] は、セキュリティ・ポリシを半自動生成するためのツールである。

通常の trace 結果に加えて、Security Context を出力するように改造された strace を用いてプログラムの挙動を解析する。次に、対話形式のインタフェースを通して type を設定する。これにより、実際に生成される前に提案されるセキュリティ・ポリシが提示される。これを手動で変更を加えば、セキュリティ・ポリシが生成される。polgen は、このように strace で得られた情報からセキュリティ・ポリシを生成するツールである。

ユーザの観点からのトップ・ダウンな判断基準ではなく、プログラムの観点からのボトム・アップな判断基準でポリシを作成する。そのため、最小特権を適切に付与できる可能性が高いが、プログラム自体に含まれている脆弱性、必要のないシステムのリソースへのアクセスを防ぐことが難しい。また現段階では、セキュリティ・ポリシの知識がなくては望まれるセキュリティ・ポリシを生成するのが難しい。

2.3.4 Virgil

Virgil[16] は GUI によるセキュリティ・ポリシの設定ツールである。その特徴はセキュリティ・ポリシに精通していない者のために作られていることである。Virgil はセキュリティ・ポリシが持つ複雑さを隠蔽して、その作成を容易にする。最小特権という観点から見ると不適切なポリシであっても、まったくポリシの存在しないサービス・アプリケーションに対するポリシを記述することでセキュリティを高めることができるという思想に基づく。そのため、生成されるセキュリティ・ポリシのセキュリティ強度は他のツールを用いたり、一から手で定義したものに比べて弱くなってしまう。

2.3.5 Simplified Policy

Simplified Policy[17] は、SELinux に於けるセキュリティ・ポリシ設定のコストを下げるために作られた中間言語である。SELinux の元々のセキュリティ・ポリシが持つ複雑さを隠蔽して、設定を容易にする目的がある。

Simplified Policy では、従来の SELinux に於けるセキュリティ・ポリシに於ける問題点を以下の用に整理している。

- リソースとラベルの関連付けが分かりにくい
- permisson 設定項目が膨大である
- 設定が散在し、設定状況が分かりにくい
- リソースにアクセス可能な domain 一覧が分かりにくい
- domain 遷移が分かりにくい
- 開発の進行による設定方法の変化

これらの問題点の解決策として考えられたのが中間言語である。また、中間言語を設定するための GUI として後述する SELinux Policy Editor[18] が作られている。中間言語であるため、問題点にも挙げられているように活発に開発が進められている SELinux においても SELinux 自体におけるポリシフォーマットの変更を気にすることなく運用を続けることができる。

具体的には、アクセス権限の指定を domain と type を使って行うのではなく、domain に対してアクセスを許可する対象を実際のパスで表記する。また、一つの domain に関係するアクセス権限の情報が全て一つのファイルに集約されているため見通しが利きやすい。permission も大幅に削減されており、Unix に於ける通常の permission と同じ r,w,x の三つと search permission にあたる s を加えた 4 つの permisson に限定することで理解を容易にしている。

SELinux のセキュリティ・ポリシと比較して容易に定義が可能であるが、ユーザに理解しやすいレベルに落とす際に permission を減らした影響で、セキュリティ強度に問

題があることが指摘されている [19]。また、システム全てのポリシを Simplified Policy で記述する必要があるため、ポリシ全てを変更しなければならない。Simplified Policy は現在も開発が継続されており、これらの問題の解決にも取り組まれている。

2.3.6 SELinux Conditional Policy Extention

SELinux Conditional Policy Extention（以後、Conditional と呼ぶ）はポリシ・ソースを変更したり、新しくセキュリティ・ポリシを読み込まずにアクセス制御の変更を行うものである。あらかじめセキュリティ・ポリシの中に、bool 変数とそれを用いた条件分岐の表現を含ませておき、SELinux の動作中に動的に bool 変数を変更することで、ポリシを変更や読み込みなおすことなくアクセス制御の変更を行う。条件分岐に置いては AND、OR、NOT、XOR などの論理演算を行うことができる。

図 2.20 のように定義する。図の例では、一行目で httpd_enable_homedirs という bool 変数を定義する。二行目以降が選択されるアクセス制御の定義である。もし、この変数が true であれば、httpd_t ドメインが user_home_dir タイプのディレクトリを開くことが許可される。このように、bool 変数を用意し、その値によってアクセス制御の変更を行うことができる。

```
bool httpd_enable_homedirs true;
if (httpd_enable_homedirs) {
allow httpd_t user_home_dir_t:dir { getattr search };
}
```

図 2.20: Conditional の定義

デフォルトのセキュリティ・ポリシを様々な環境に適用させることを目的とする。bool 変数の例として httpd_enable_cgi が挙げられる。これは、Apache において、CGI の利用を許可するための bool 変数である。これは setsebool コマンドを用いるか、booleans ファイルに変更を加えることで変更することができる。これにより、ポリシ・ソース自体を書き換えることなく、適用するアクセス制御を変更することができる。

しかし、あるラベルが貼り付けられるべき実際のパスは無数に存在する。このような場合には Conditional を用いることは難しい。そのため、Conditional だけでは、ポリシ・ソース自体に変更を加えずに、全ての環境に適用することはできない。

2.4 SELinux の現状

現在、SELinux の開発は開発元である NSA と、Linux のディストリビューションの一つである Fedora Core Linux を管理・製作している RedHat[20]、Tresys[21] の三団体が中心となって進められている。SELinux の開発に於ける実験的な要素や標準のセキュ

リティ・ポリシは Fedora Core Linux で実証実験が行われている。現在では、Fedora Core Linux4 はインストール後、標準で SELinux が targeted ポリシで動作する。

また、Fedora Core 5 test-1 ではセキュリティ・ポリシのソースをユーザから隠蔽する試みが行われている。コンパイル後のセキュリティ・ポリシ・バイナリだけを提供する試みである。これは、セキュリティ・ポリシを SELinux に精通していない管理者などのユーザに定義させるのではなく、サービスや SELinux に精通した人間が書いたセキュリティ・ポリシを広く使うべきであるという考えに基づいている。アクセス制御の細かな変更は Conditional（第 2.3.6 節参照）を用いている。しかし、パスなどの具体的な値が Fedora Core Linux の環境に大きく依存している。そのため、セキュリティ・ポリシ・ソースを取得せずにホストに合わせたアクセス制御を行うのは不可能である。

第3章　問題点の整理

本章では、SELinux の導入・運用コストを増大させているセキュリティ・ポリシの問題について整理する。セキュリティ・ポリシの難解さについて四つの観点から述べる。

3.1　定義方法の理解の難解さ

第 2.2 節で述べたように、セキュリティ・ポリシの定義には様々な定義項目が存在する。開発中のものを含めれば、アクセス制御だけで TE、RBAC、MLS、MCS、Constraint の合計 5 つの機能がある。

例として、第 2.2.2 節で述べた TE に関するポリシ定義であるアクセスベクタを取り上げる。アクセスベクタの定義を行う際には、あらかじめ各オブジェクトクラスに応じて用意された permission の中からアクセス権限として許可するものを選択する必要がある。しかし、このオブジェクトクラスと permission の組み合わせの数は非常に多い。具体的な数字を挙げれば、ファイル関係のオブジェクトクラスに限っても、file, blk_file, chr_file, fifo_file, lnk_file, sock_file, dir と 7 種類、permission は ioctl, read, write, append, swapon など 16 種類（特定のオブジェクトクラスにだけ存在する permission を含めればそれ以上）存在する [22]。これらの個々のオブジェクトクラスや permission を理解するためには OS に関する深い知識が必要である。また、全てのオブジェクトクラスと permission を組み合わせると、その総数は 100 種類を超える。

このように、セキュリティ・ポリシの定義フォーマットや値の意味などの定義方法を理解することは難しい。

3.2　ポリシの見通しの悪さ

SELinux は非常に粒度の細かいアクセス制御を行うことができる。しかし一方で、許可する全てのアクセスをセキュリティ・ポリシとして定義する必要がある。そのため、セキュリティ・ポリシは非常に冗長で膨大になる。具体的な例としては、Fedora Core Linux 4 に付属しているデフォルトのセキュリティ・ポリシは、コメント行と空行を削除した状態で、strict のセキュリティ・ポリシは 62284 行、容量にして 3.8MB 程度、targeted のセキュリティ・ポリシでも 23839 行、容量にして 1.4MB 程度になる。

また、第 2.2.4 節で述べたマクロを使って同様の記述はまとめて記述できる。しかし、マクロ自体も入り組んでいる。そのため、マクロを用いることはセキュリティ・ポリシ全体の見通しを悪化させることにも繋がる。

Attribute を用いてグループ化された権限を付与されている場合も同様である。関連付けられている Attribute が、どのような権限を持っているかを逐次確認する必要がある。

第 3.1 で述べたように定義方法の理解の難しさも相まって、定義されたセキュリティ・ポリシの内容を把握することは難しい。

3.3 ポリシの定義の困難さ

セキュリティ・ポリシを定義することは難しい。なぜなら、セキュリティ・ポリシは全ての許可されるアクセスについて定義する必要があり、サービス・アプリケーションがどのようなリソースに対してアクセスを行うかといったソフトウェアの動作やシステム側の挙動に精通している必要があるからである。

日常的に利用しているサービス・アプリケーションであっても、全てのアクセスを細かに把握することはできない。そのため、セキュリティ・ポリシを一から定義するのは単にサービス・アプリケーションに精通しているだけでは難しい。このように、現実的にセキュリティ・ポリシに精通しているものではなくては、十分セキュリティ強度を持ったセキュリティ・ポリシを定義することはできない。

また、セキュリティ・ポリシを定義する際には、依存性の点にも考慮しなければならない。これは一つのリソースに対して付与できる Security Context は一つだけであり、アクセス制御は付与された Security Context に基づいて行われるためである。Apache や Webalizer[23] のような非常に密接な関係にあるサービス・アプリケーション間の依存性は言うまでもなく、特定のサービス・アプリケーションしか利用しないと思われたファイルを別のサービス・アプリケーションが利用するといった思いもよらない場所で依存性の問題が起こる可能性がある。

3.4 ポリシの変更の困難さ

標準のセキュリティ・ポリシの開発は、Fedora Core Linux にフィードバックする形で行われている。そのため、標準のセキュリティ・ポリシは Fedora Core Linux の環境に強く依存する形になっている。

サービス・アプリケーションはどのような環境であっても、設定が同じであれば同様に動作する。また、たとえ設定を変更したとしても、1 からアクセス制御を書き直さなければならないほど動作が変化するものではない。すなわち、TE の定義という観点から見ればセキュリティ・ポリシにはある程度の再利用性があると言える。しかし、標準のセキュリティ・ポリシに於けるアクセス権限は必要最低限、もしくは、そのセキュリティ・ポリシを作成した人間の利用の範囲に限定されているため、設定を変更すると正常に動作しない場合がある。このような場合にはセキュリティ・ポリシの変更が必要である。

しかし、セキュリティ・ポリシを変更することは難しい。なぜなら、あるラベルがどのようなリソースに付与されるべきラベルであり、どのような権限を持つラベルなのかを知るためには、変更を行うセキュリティ・ポリシを理解しなければならないからである。また、変更を行った際に正しく SELinux が動作しなかった場合などにもセキュリティ・ポリシの知識が必要になる。

第4章　本研究の手法

本章では、第3章で挙げた問題点を解決するためにセキュリティ・ポリシ定義の自動化を行うアプローチについて議論する。

4.1　概要

本研究では、ベースとなるセキュリティ・ポリシをホストに合わせて自動で変更する。セキュリティ・ポリシを自動生成する手法を提案する。

既存の手法を用いて、セキュリティ・ポリシを一から自動生成するのには様々な問題がある。例えば、自動生成を正しく行うためにセキュリティ・ポリシに関する知識が必要であることなどである。また、一から自動生成したセキュリティ・ポリシの正当性は保証されない。すなわち、自動生成されたセキュリティ・ポリシが意図したアクセス制御を行うかどうか保証されない。自動生成されたセキュリティ・ポリシが意図したアクセス制御を行うかどうかは利用者自身が判断する必要がある。SELinux の問題点はセキュリティ・ポリシの定義が難解さである。また、その難解さの理由にセキュリティ・ポリシ自体を理解することの難しさが含まれることは第3章で述べた。その点で、既存の自動生成手法は、SELinux の問題を解決していない。SELinux の問題を解決するためには、セキュリティ・ポリシに関する知識を持たない者でも、セキュリティ・ポリシが定義出来る必要がある。

そこで本研究では、セキュリティ・ポリシの自動生成を行う手法として、あらかじめ定義されたセキュリティ・ポリシの変更を自動化する。対象のサービス・アプリケーションやホストの構成を調べることでセキュリティ・ポリシを自動的に変更する。また、自動生成のベースとなるセキュリティ・ポリシは SELinux に精通した者が定義したものを用いる。これにより、高いセキュリティ強度を保ちながら、SELinux の導入・運用コストの削減が可能である。

本研究においては、前提とする知識を必要としないセキュリティ・ポリシの自動生成を実現することで、これらのセキュリティ・ポリシの問題の解決を試みる。

本手法を実現するためには、セキュリティ・ポリシの再利用について考慮する必要がある。そこで、セキュリティ・ポリシ再利用のためのフレームワークを提案する。以下に、実現方法として具体的なセキュリティ・ポリシの再利用と調整の方法について整理し、フレームワークについて述べる。また、その運用モデルについて述べる。

4.2 再利用と調整

第 2.1 節で述べたように、SELinux のアクセス制御機能は TE と RBAC からなる。同様に、セキュリティ・ポリシも TE の定義と RBAC の定義に分けることができる。

本研究では RBAC の定義に関する自動化は行わず、TE の定義に注目して自動化を行う。これには二つの理由がある。一つは、どの利用者にどのような権限を与えるかという組み合わせは無数に存在する。そのうち、どれが適切でどれが適切でないかは判断できない。そのため、利用者にどのような権限を与えるかは自動化できないからである。もう一つは、セキュリティ・ポリシの難解さは TE の定義が難解であることが主な原因であるからである。

TE の定義は大きく分けると二つに分類される。ラベルによって抽象化されたアクセス制御の定義と、実際にラベルをリソースに対して付与する定義である。

アクセス制御はプロセスからリソースに対するアクセスを、それぞれに付与されたラベルを用いて定義する。Apache であれば、httpd_t ドメインが付与されたプロセスが httpd_sys_content_t タイプを付与されたファイルを読み込むことや、httpd_t ドメインの付与されたプロセスが http_port_t タイプの付与されたポート番号を Listen するということが定義されている。このように、アクセス制御の定義には、httpd_sys_content_t タイプというラベルが実際にどのファイルに付与されるかは定義されない。アクセス制御はラベルによって抽象化されている。したがって、アクセス制御の定義は、環境が変わっても同様に適用することができる。

ただし、再利用を行う際には、適用するアクセス制御の定義を選択しなければならない。再利用元にするセキュリティ・ポリシで必要とされているアクセス権限の全てが再利用先のホストに必要とは限らないからである。必要な動作を行うことができる範囲内であれば、与えるアクセス権限は少なければ少ないほど良い。例として、Apache における tty を開く権限を持つためのアクセス制御の定義を挙げる。このアクセス権限は、SSL 利用のためにパスフレーズの設定された秘密鍵を用いる際に必要とされる。しかし、パスフレーズの設定された秘密鍵を用いない場合にはこの権限は不要である。したがって、アクセス制御の定義の再利用を行う際には、適用するアクセス制御の定義を選択しなければならない。

逆にアクセス制御に用いられるラベルを実際に付与する定義には再利用性がない。ラベルと実際のパスの組み合わせは無数に存在するためである。例として、Apache において DocumentRoot に付与される httpd_sys_content_t タイプというラベルについて取りあげる。ソースからコンパイルした場合のデフォルトの DocumentRoot は /usr/local/apache/htdocs である。しかし、あるディストリビューションのパッケージ化された Apache の DocumentRoot が/usr/local/apache/htdocs に設定される保証はない。また、DocumenRoot がどこに設定されるかはユーザによって変更される可能性もある。このようにラベルと実際のパスの組み合わせは無数に存在する。そのため、ラベルを実際に付与する定義は再利用できない。セキュリティ・ポリシを再利用する際には、ラベルを実際に付与する定義は全て定義しなおす必要がある。

4.3 フレームワーク

本研究におけるフレームワークは、セキュリティ・ポリシの専門家である者（以後、ポリシメンテナと呼ぶ）と、ポリシメンテナによって書かれるセキュリティ・ポリシのベース (以後、ポリシ・テンプレートと呼ぶ）とそれをホストに合わせて変更するスクリプト (以後、調整スクリプトと呼ぶ)、それらを利用して自分のホストのセキュリティ・ポリシの生成を行う者（以後、ユーザと呼ぶ）から構成される。

まず、本フレームワークにおける調整スクリプトの実装方法を定める。調整スクリプトは Unix の標準的なシェルである Bourne Shell に準拠した ShellScript を用いて実装を行うものとする。これは環境への依存性を下げるためである。

本フレームワークでは、セキュリティ・ポリシの定義を行うサービス・アプリケーションの設定ファイルから情報を抽出する。また、そのサービス・アプリケーションに関わるリソースの実際のパスといったシステムの構成について情報を抽出する。これらの情報を用いて、セキュリティ・ポリシの変更を調整し、ホストに応じたセキュリティ・ポリシを自動生成する。しかし、設定ファイルはサービス・アプリケーションによって異なる。また、必要なシステム構成の情報も異なる。そのため、調整スクリプトはサービス・アプリケーション毎に一から記述する必要がある。したがって、その作成コストはポリシメンテナにとって非常に大きなものになってしまう。そこで少しでもこのコストを緩和するために、本フレームワークでは統一したフォーマットを定める。統一したフォーマットを用いることで、作者以外でもメンテナンスを容易に行うことができる。また、このフォーマットには次のような要件が求められる。

- ポリシの変更の簡易化
- ポリシ・テンプレートと調整スクリプトの依存性の解決

ホストに合わせてセキュリティ・ポリシを再利用するためには、適用するアクセス制御の定義の選択とラベルを実際に付与する定義の記述が必要である。これらの変更をより容易に行うためのフォーマットを定義する必要がある。

また、SELinux のセキュリティ・ポリシには、先行研究 [24][17] のような幾つかのポリシ・フォーマットとポリシ・タイプが存在する。これらの全てに適用出来るようにするため、調整スクリプトがポリシ・テンプレートに依存しないようにフォーマットを定める必要がある。これらのフォーマットとタイプはそれぞれ異なった目的によって存在するため、どのフォーマットとタイプが最も良いということはない。したがって、本フレームワークは、いずれのフォーマットとタイプでも動作する必要がある。調整スクリプトとポリシ・テンプレートの依存性を解決することで、いずれかのフォーマットとタイプのポリシ・テンプレート向けに書かれた調整スクリプトを、別のポリシ・テンプレートにも用いることができる。これらの関係を図 4.1 に示す。フォーマットとタイプは階層構造を用いて整理することができる。図の”te.in/fc.in”の部分がポリシ・テンプレートである。ポリシ・テンプレートはフォーマットとタイプ毎に必要である。しかし、調整スクリプトはいずれのポリシ・テンプレートであっても利用できる。

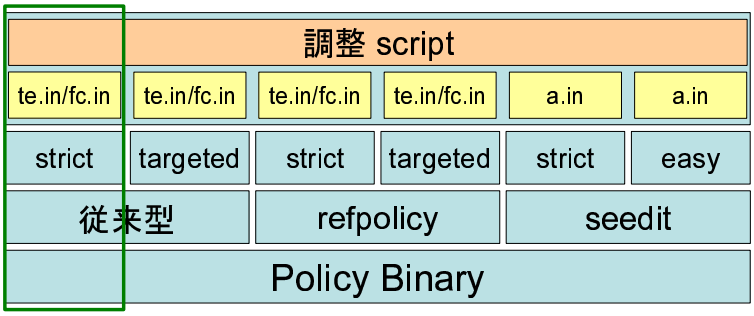

図 4.1: 各種ポリシ・フォーマットとフレームワークの関係

これらの要件を考慮した上で、フォーマットを定義する。前述したように、セキュリティ・ポリシを再利用するためには、適用するアクセス制御の定義の選択とラベルを実際に付与する定義の記述が必要である。

適用するアクセス制御の選択は Conditional（第 2.3.6 節参照）を用いる。Conditional はあらかじめセキュリティ・ポリシに定義された bool 変数を用いて、適用するアクセス制御を変更する。調整スクリプトはこの bool 変数の値を変更する。これにより、特別なフォーマットを定義することなく変更が可能である。

ラベルを実際に付与する定義は全て記述する必要がある。あるラベルが付与されるべき対象は、変更を行うサービス・アプリケーションのリソースとして抽象化された名称で呼ぶことができる。Apache であれば、公開される静的なコンテンツは DocumentRoot と呼ぶことが出来る。本フレームワークでは、この名称を”@”で囲んで、置換を行うための変数とする。

また、調整スクリプトがポリシ・テンプレートに加える変更は bool 変数と”@”で囲まれた変数の変更のみに限定する。これにより、調整スクリプトが行う変更は値の代入だけになるため、特定のポリシ・フォーマットに依存することがなくなる。

以上から、ポリシ・テンプレートと調整スクリプトの関係を図 4.2 のように定める。ポリシ・テンプレートは、アクセス制御の定義を行う.te ファイルの元となる.te.in とラベルの定義を行う.fc ファイルの元となる.fc.in からなる。調整スクリプトはシステムの構成とアプリケーションの設定ファイルを参照する。ここから得られた情報を用いて、ポリシ・テンプレートの中から bool 変数と”@”で囲まれた変数を変更することでセキュリティ・ポリシを自動生成する。

4.4 運用モデル

本研究では運用モデルとして、情報共有・流通型のモデルを提案する。SELinux とセキュリティ・ポリシに精通している者の知識を共有することで、SELinux の導入のコストを削減する。

現在では図 4.3 のように、特定環境・用途向けのセキュリティ・ポリシをデフォルトのセキュリティ・ポリシとして管理・配布する流通手法が存在する。第 2.3.5 節で述べた

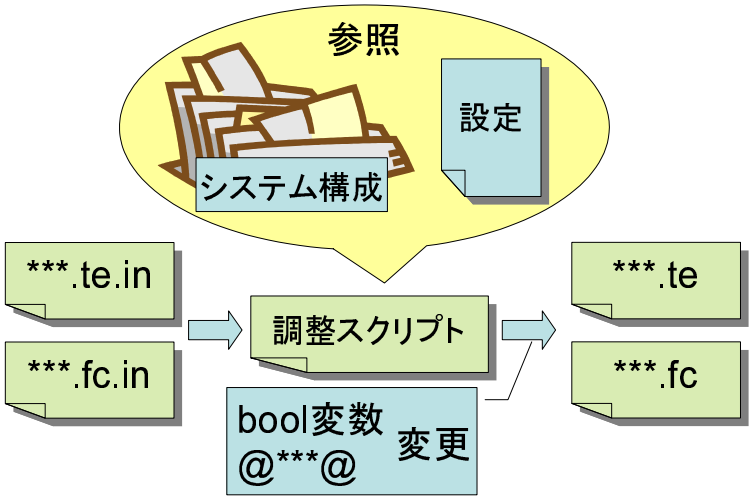

図 4.2: ポリシ・テンプレートと調整スクリプトの関係

Simplified Policy のようにポリシ・フォーマットのプロジェクト単位でセキュリティ・ポリシが管理・配布されている。

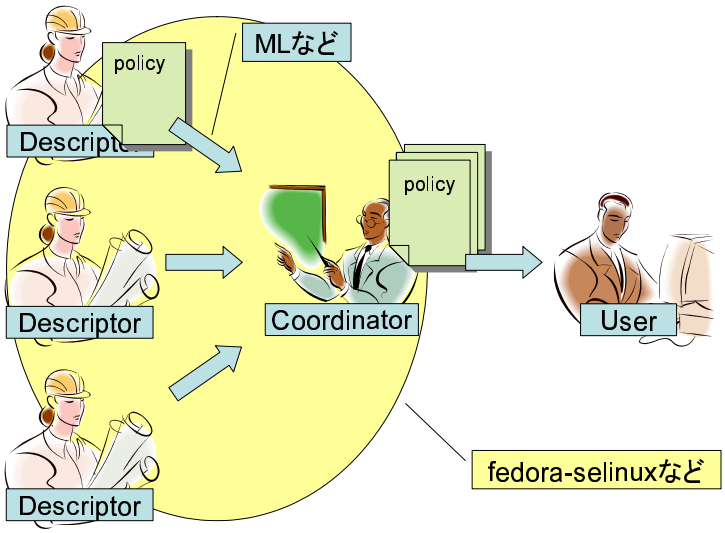

図 4.3: セキュリティ・ポリシの流通手法

しかし、デフォルトのセキュリティ・ポリシを配布する方法では、ラベルを付与する実際のパスの違いなどからディストリビューション毎に分割管理する必要がある。そのため、プロジェクトに加えてディストリビューション毎に管理・配布されている。現在のセキュリティ・ポリシ分割管理を図 4.4 に示す。

本研究では既存の流通手法に則り、セキュリティ・ポリシのベースとそれを自動調整するためのスクリプトをポリシメンテナが記述し配布する手法を用いる。これにより、SELinux に精通していない人でも、各々のホストに合わせてセキュリティ・ポリシを自

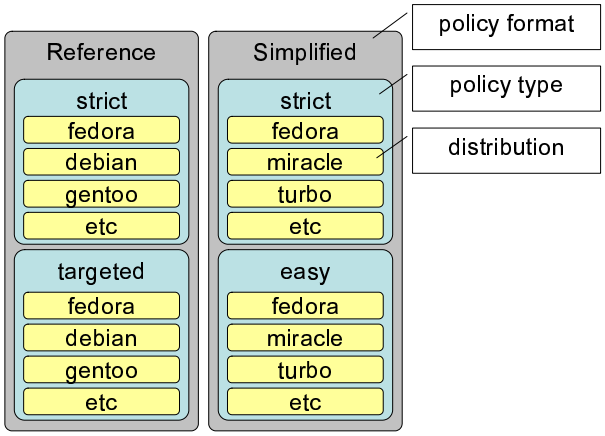

図 4.4: ポリシの分割管理

動生成することができる。また、環境が異なるディストリビューション毎にセキュリティ・ポリシを分割管理する必要がなくなる。これは環境に合わせてセキュリティ・ポリシの変更を行うコストを削減させることに繋がる。本研究で提案する運用モデルを図 4.5 に示す。

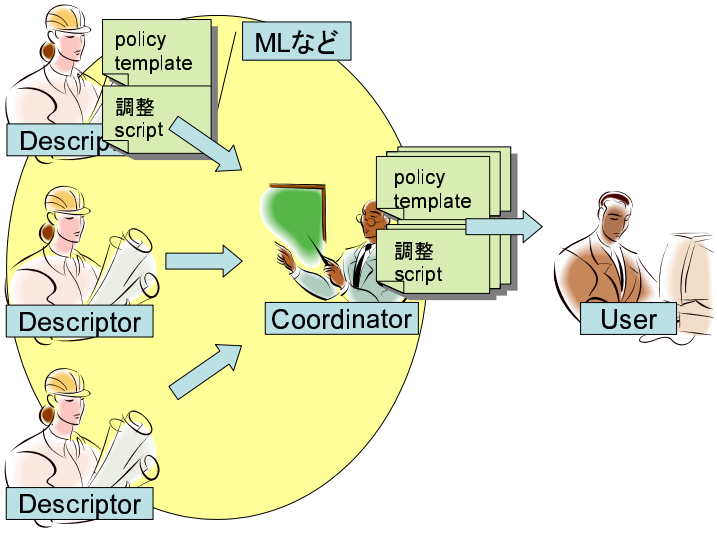

図 4.5: 本研究の提案する流通モデル

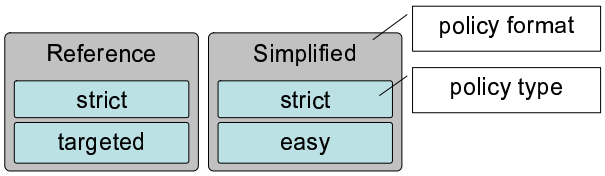

図 4.6: 本研究のポリシ分割管理モデル

第5章　実装

本研究では、システム構成とアプリケーション設定に基づく自動生成フレームワークの提案を行った。本章では、実例として Apache のセキュリティ・ポリシを自動生成するツールの実装を行う。

5.1　概要

本ツールは、セキュリティ・ポリシの雛形となるポリシテンプレート (.fc.in/.te.in) と調整スクリプト (conf) から成る。

あるサービス・アプリケーションのセキュリティ・ポリシを本ツールによる自動生成で導入する場合には、そのサービス・アプリケーション用のポリシテンプレートと調整スクリプトが必要である。これらのファイルは、SELinux とそのサービス・アプリケーションに精通した人間によって書かれ、配布される。

本章では、実例として Apache のセキュリティ・ポリシを自動生成するツールの実装を行った。

5.2　動作例

Apache のセキュリティ・ポリシを自動生成する動作例を挙げる。Apache のセキュリティ・ポリシを生成するために必要なファイルは、ポリシーの雛形となるポリシテンプレートの apache.fc.in と apache.te.in、調整スクリプトの conf である。ユーザは Web 経由などでこれらのファイルを入手する必要がある。ファイルを入手したら”./conf”コマンドを実行する。

図 5.1 のように、apache.fc と apache.te という 2 つのファイルを自動生成する。これらのファイルが Apache のセキュリティ・ポリシである。

apache.te.in は apache.te の雛形となるファイルである。apache.te はラベルを用いたアクセス制御の定義を行うためのものである。本ツールではアクセス制御の定義の一つである bool 変数を変更する。そのため、apache.te と apache.te.in の内容は同じままである。

apache.fc.in は apache.fc の雛形となるファイルである。apache.fc は、Security Context を実際のパスへ付与する定義を行うものである。apache.fc.in は、実際のパスを変数に置き換えたものである。

```
$ ls
apache.fc.in  apache.te.in  conf
$ ./conf    <=== conf を立ち上げる
checking for binary ... /usr/sbin/httpd existed.
checking for apachectl ... /usr/sbin/apachectl existed.
checking for httpd -V ... done.
checking for server root ... /etc/httpd existed.
checking for suexec ... /usr/sbin/suexec existed.
checking for httpd.conf ... /etc/httpd/conf/httpd.conf existed.
parsing for httpd.conf ................. done.
calculating path of APACHE_BIN ... done.
（略）
autogenarate done.
$ ls
apache.fc  apache.fc.in  apache.te  apache.te.in conf
```

図 5.1: 自動生成ツールの動作例

デフォルトのセキュリティ・ポリシは想定される環境に基づいて定義されている。Apache の実行バイナリである httpd が/usr/local/sbin 以下にあることを想定する。その場合、apache.fc の中には図 5.2 のような一文が含まれる。

```
/usr/local/sbin/httpd  --  system_u:object_r:httpd_exec_t
```

図 5.2: デフォルトの apache.fc の一例

しかし実際には、Apache の実行バイナリが何処にインストールされるかは環境に依存する。Fedora Core Linux であれば/usr/sbin 以下である。したがって、このセキュリティ・ポリシは適用することができない。また、Apache のソース配布直後の設定では/usr/local/apache 以下に全てのファイルがインストールされる。このように、ファイルパスを直接指定することは環境に強く依存する。結果として、移植性を欠いてしまうことに繋がる。そこで、図 5.3 のように、フレームワークの設計（第 4.3 節参照）に基づいて、ファイルパスを指定する部分を変数に置き換える。このようにして置き換えたものがポリシ・テンプレートの apache.fc.in である。

調整スクリプトはホストの環境を調査する。具体的には、想定されるパスの中から Apache の実行バイナリを検索する。そして位置が特定できれば、@APACHE_BIN@を実際のファイルパスへと置き換える。Fedora Core Linux であれば図 5.4 のように置換されるはずである。

```
@APACHE_BIN@       --  system_u:object_r:httpd_exec_t
```

図 5.3: apach.fc.in の一例

```
/usr/sbin/httpd  --  system_u:object_r:httpd_exec_t
```

図 5.4: 自動生成された apache.fc の一例

想定されるパスの中に Apache の実行バイナリを発見することができなかった場合は、調整スクリプトは終了する。このような場合には、ユーザは引数としてパスを与えることで自動生成を行うことができる。

このように、調整スクリプトはポリシテンプレートをホストの環境に合わせて調整し、セキュリティ・ポリシを自動生成する。

5.3 実装環境

調整スクリプトは、フレームワークに準じて Unix の標準的なシェルである Bourne Shell に準拠した Shell Script を用いて実装を行った。

セキュリティ・ポリシを定義する対象として Apache を用いる。表 5.1 に、実装に用いたソフトウェア環境を示す。

表 5.1: 実装ソフトウェア環境

OS	Fedora Core Linux 4 (Kernel 2.6.14-1)
ソフトウェア	Apache 2.0.54 GNU bash 3.0.16 GNU sed 4.04
セキュリティ・ポリシ	strict-1.27.1-2.16

5.4 設計

本節では Apache のセキュリティ・ポリシを自動生成する調整スクリプトとポリシ・テンプレートの設計を行う。

設計を行うために、Apache の動作概要と Apache に関するリソースについて整理を行う。Apache に関するデフォルトのセキュリティ・ポリシをアクセス制御を Security Context について整理する。

5.4.1 Apache に関する整理

Apache の動作を整理すると、以下のようになる。

- http サービスのデーモンである
- httpd もしくは apache、apache2 という名前のバイナリを実体とする
- 設定ファイルのパスはコンパイル時に指定される
- -V オプションでコンパイル時のオプションを確認できる
- -f オプションを用いて起動することで利用する設定ファイルを上書きできる
- 設定ファイルに記載することで別の設定ファイルを読み込むことができる
- デフォルトの設定ファイルは httpd.conf である

Apache が利用するリソースに関する情報はコンパイル時か設定ファイルの中で記述される。”httpd -V”コマンドを実行することでコンパイル時に指定されたオプションが閲覧可能である。したがって、Apache のリソースに関する情報は、実行バイナリと設定ファイルの二つから得られる。

Apache に関するリソースと、設定ファイルと”httpd -V”コマンドから得られる情報を表 5.2 に整理する。Apache に関するリソースの中で設定ファイルなどから情報を得られないのは、実行バイナリと設定ファイルである。設定ファイルはコンパイル時に指定されている。しかし、起動時に-f オプションを用いて、別の設定ファイルを読み込むことができる。したがって、設定ファイルも情報が得られない可能性がある。

5.4.2 セキュリティ・ポリシに関する整理

セキュリティ・ポリシを定義するために、ある domain がある type に対して行うアクセスを厳密に整理する必要がある。そのために、サービス・アプリケーションが利用するリソースの整理が必要である。

アクセス制御はプロセスやファイルなどに付与された type, domain などの Security Context を用いて行う。そのため、リソースは Security Context を用いて整理される必要がある。

Security Context は伝統的に命名規則が定められている。サービスの domain_リソースの分類_t と名づけられるのが一般的である。デフォルトのセキュリティ・ポリシにおいて、Apache（httpd）が用いるリソースは主に次のように分類される。

httpd_t

Apache 用の domain である。Apache に関するアクセス制御はこの domain に基づいて行われる。

表 5.2: Apache Resource

リソース	情報の取得元
実行バイナリ (httpd)	なし
設定ファイル (httpd.conf)	なし/"httpd -V"/Include ディレクティブ
ベースディレクトリ (ex:/usr/local/apache)	ServerRoot ディレクティブ
html などのウェブコンテンツ	DocumentRoot ディレクティブ
CGI スクリプト (ex:ServerRoot/cgi-bin)	ScriptAlias ディレクティブ
ユーザディレクトリ (ex:public_html)	UserDir ディレクティブ
エラーログ (ex:ServerRoot/logs/error_log)	ErrorLog ディレクティブ
アクセスログなど	CustomLog ディレクティブ
スクリプトログ	ScriptLog ディレクティブ
モジュール (ex:ServerRoot/modules)	LoadModule ディレクティブ
キャッシュファイル用ディレクトリ	CacheRoot ディレクティブ
プロセス ID 記録ファイル	PidFile ディレクティブ
ロックファイル	LockFile ディレクティブ
ポート番号	Listen ディレクティブ

httpd_exec_t

Apache の実行バイナリ用の type である。domain 遷移のエントリポイントでもある。この type のファイルが initrc_t ドメインによって実行されることで httpd_t ドメインへと遷移する。

httpd_config_t

Apache のコンフィグファイル用の type である。httpd_t ドメインから読み込む権限がある。httpd_t ドメインから write や append の permisson は許可しない。後述する httpd_*_content_t タイプとの明確な差異は、Webalizer など、Apache 以外のサービス・アプリケーションからのアクセスが基本的に許可されないことである。

httpd_*1_content_t

html などの静的な参照が行われるリソース用の type である。どの user 属性に所属しているかによって*1 の部分が変更される。DocumentRoot 化に置かれている場合、特に httpd_sys_content_t タイプとする。

httpd_t ドメインから読み込む権限がある。基本的に httod_config_t タイプと同じく、write や append は許可しない。ただし、SELinux は特定の type のファイルの exec システムコールをきっかけにして domain を遷移する。そのため、php の

ような、新たなプロセスを作ることなく Apache プロセス内で処理を行うスクリプト言語を用いる場合に問題がある。php を利用する場合は、httpd_t ドメインに httpd_*_content_t タイプを実行する権限を与えるものとする。

httpd_*1_script_t

Apache から CGI として呼び出されるスクリプト・プログラム用の domain である。httpd_t ドメインから httpd_*_script_exec_t タイプをエントリポイントとして遷移する。httpd_*_content_t タイプと同じく、どの user 属性に所属しているかを区別する。これにより、他の user 属性に所属している domain から保護する。

httpd_*1_script_exec_t

CGI として呼び出されるスクリプト・プログラムのファイルに付与される type である。

httpd_*1_script_*2_t

CGI で利用されるファイルのための type である。httpd_*1_script_t ドメインは、静的なコンテンツに付与されるラベル httpd_*1_content_t タイプに対して何のアクセス権限も持たない。CGI に利用するためのファイルは明示的に宣言をする必要がある。*2 に設定する文字列によって適切なアクセス権限を設定することができる。*2 に入れることができるのは”ro,ra,rw”の三つである。それぞれ、ReadOnly、Read/Append、Read/Write の権限を持つ。

httpd_log_t

ログファイルのための type である。httpd_t ドメインや httpd_*_script_t ドメインなど、Apache 関連の多くの domain からアクセスを許可する。ただし、アクセスの内容は read もしくは append であり、write は許可されない。

httpd_module_t

モジュールのための type である。httpd_t ドメインからの実行権限が与えられる。

httpd_var_run_t

/var/run 以下に置かれる Apache 関連のファイルのための type である。

http_port_t

port 番号 80、443 に対して付与される type である。

Security Context とそれにに付与されるアクセス権限が正しく分類されていても、正しくリソースに付与できなければ意味がない。httpd_t ドメインは httpd_sys_content_t タイプと httpd_config_t タイプに対して、ほぼ同じアクセス権限を持つ。しかし、httpd_t ドメイン以外からのアクセスという観点では大きな差異がある。そのため、この付与

を正しく行われなかった場合、悪意を持った domain からの設定情報へのアクセスを許してしまう危険性がある。

正しく Security Context を付与するためには、システム上のリソースを正しく分類する必要がある。

5.4.3 リソースとラベル

本ツールは、httpd.conf とコンパイル時に指定されたオプションを用いて、Apache が利用するリソースを抽出し整理する。コンパイル時に指定されたオプションは、設定ファイルで明示的に定義がなかった場合にデフォルト設定として用いる。

Apache の設定ファイルの中からディレクティブ情報として得られたリソースの分類と SELinux の Security Context の対応を整理すると、以下 5.3 の表のようになる。

表 5.3: Apache Resource

リソース	ディレクティブ	Security Context
実行バイナリ (httpd)	なし	httpd_exec_t
設定ファイル (httpd.conf)	なし/Include	httpd_conf_t
ベースディレクトリ (ex:/usr/local/apache)	ServerRoot	httpd_conf_t
ウェブ上のドキュメントツリー	DocumentRoot	httpd_sys_content_t
CGI スクリプト (ex:ServerRoot/cgi-bin)	ScriptAlias	httpd_sys_script_t
ユーザディレクトリ (ex:public_html)	UserDir	httpd_USER_content_t
エラーログ (ex:ServerRoot/logs/error_log)	ErrorLog	httpd_log_t
アクセスログなど	CustomLog	httpd_log_t
スクリプトログ	ScriptLog	httpd_log_t
モジュール (ex:ServerRoot/modules)	LoadModule	httpd_module_t
キャッシュファイル用ディレクトリ	CacheRoot	httpd_cache_t
プロセス ID 記録ファイル	PidFile	httpd_var_run_t
ロックファイル	LockFile	httpd_var_run_t
port 番号	Listen	httpd_port_t

5.5 実装

本節では、Apache のセキュリティ・ポリシの自動生成を実装した。ポリシテンプレートと調整スクリプトを説明する。

5.5.1 ポリシテンプレート概要

ポリシテンプレートは Security Context の付与を行う.fc ファイルと、TE に関わる定義の書かれた.te ファイルからなる。

Apache の標準セキュリティ・ポリシは長い時間をかけて様々な環境に適応するように作られてきた。そのため、Apache の標準セキュリティ・ポリシには以下の 2 つの特徴がある。

- TE に関わる定義が非常に洗練されている
- Security Context の付与の定義がつぎはぎになっている

第 2.2.2 節で述べたように SELinux は Security Context というラベルを用いてアクセス制御を行う。そのため、Security Context から Security Context へのアクセス権限というように抽象化され、TE は洗練されてきた。第 2.3.6 節で述べた Conditional も 7 種類設定されており、自身の環境に合わせて容易に変更することができる。

最も洗練されているところは、多くの bool 変数を用意し、TE に関する定義の選択肢を増やしたことではない。むしろ、bool 変数は最小限に留めて、Security Context をどのように付与するかということでアクセス制御を行う点にある。TE に関する定義はある domain を持ったサブジェクトからある type をもったオブジェクトに対して、どのアクセスを許可するかを定義したものである。実際に、どのプロセスにその domain が付与されるかは Security Context が付与されなければ分からない。同様に Apache の例であれば、TE に関する定義で cgi の利用を許可しても、それだけだけでは動作しない。実際に cgi を動作させるためには、そのために用意された cgi 用の domain と type を正しく付与する必要がある。また、付与をどのように行うかを工夫することで、より正確に管理者の意図をアクセス制御に反映することができる。

しかし、実際のホスト上のファイルパスを Security Context に付与する定義は様々な環境に適応するために非常に複雑になっている。DocumentRoot にあたるファイルパスが複数定義されているなど、非常に対症療法的な対応になっている。

確かに、SELinux では存在しないファイルパスに対して Security Context が宣言されていても、問題を生じない。しかし、そのように多くの存在しないファイルパスが宣言を続けるのは望ましくない。同一のファイルパスに対して別の Security Context を定義するなど、定義がコンフリクトを起こす可能性がある。

また、標準のセキュリティ・ポリシに宣言されていない環境で用いようとした場合、変更が難しい。これは、Security Context の定義が複雑になった結果、サービス・アプリケーションのどのリソースが、どの Security Context であるべきかという可読性を損なったからである。

5.5.2 ポリシテンプレートの具体例

本研究では、TE に関わる定義は標準のセキュリティ・ポリシをそのまま利用する。調整は、対象ホストのシステム構成やアプリケーション設定から、表 5.4 の bool 変数

を適切な値に変更することで行う。

表 5.4: apache.te における bool 変数

bool 変数名	説明
httpd_unified	アクセス制御を緩める 全ての httpd_content が read/write/excute の権限を持つ
httpd_builtin_scripting	PHP の利用を許可する
httpd_enable_cgi	CGI の利用を許可する
httpd_enable_homedirs	/home/*/public_htm 以下での www の公開を許可する
httpd_ssi_exec	SSI を動作させるために必要
httpd_tty_comm	SSL での通信に関わる 起動時秘密鍵の passphrase を入力を求める場合に必要
httpd_can_network_connect	httpd_t が listen ではなく connect するのを許可する Reverse Proxy などに必要

また、新たに Security Context に関わる定義を作成し、第 5.4.3 節で行った整理を元に調整を行うものとする。新たに定義した Security Context は図 5.5 のようになる。

従来の Secutity Context の宣言は、単純にあるファイルパスに対して付与する Security Context を書くことが多かった。本研究では、自動生成を行う処理を通すため、どのようなリソースであるかを明示的に示した上で Security Context を記述することができる。図 5.5 の例であれば、@SERVER_ROOT@すなわちディレクティブにおける ServerRoot に該当するリソース (ディレクトリ) に対して httpd_config_t を付与することが分かる。

5.5.3 調整スクリプト

調整スクリプトはシステム構成とアプリケーション設定を調査する。その後、ポリシテンプレートからセキュリティ・ポリシを自動生成する。本研究では Shell Script で実装を行った。

フレームワークの設計に準じて、調整スクリプトは以下の二種類の出力を行う。

- 適切なファイルパスに Security Context を定義
- bool 変数への適切な値の定義

どのリソースにどの Security Context の付与すべきかという考え方は第 5.4.3 節で行った整理をそのまま用いることができる。対応するディレクティブが存在するリソースは、対応するディレクティブに設定されたファイルパスを Security Context の付与にも用いればよい。あるディレクティブに代表されるようなリソースは対応する Security Context の部分集合であると言えるからである。

bool 変数の代入は表 5.5 の指針に従うものとする。

```
# binary as follows
@APACHE_BIN@        --  system_u:object_r:httpd_exec_t
@APACHECTL_BIN@     --  system_u:object_r:httpd_initrc_exec_t
@SUEXEC_BIN@        --  system_u:object_r:httpd_suexec_exec_t

# config as follows
@SERVER_ROOT@       -d  system_u:object_r:httpd_config_t
@HTTPD_CONF@        --  system_u:object_r:httpd_config_t
@INCLUDE@               system_u:object_r:httpd_config_t

# logs as follows
@LOG_DIR@           -d  system_u:object_r:httpd_log_t
@SERVER_ROOT@/logs      system_u:object_r:httpd_log_t
@ERROR_LOG@         --  system_u:object_r:httpd_log_t

@LOAD_MODULE        --  system_u:object_r:httpd_module_t
（略）
```

図 5.5: apache.fc.in

表 5.5: bool 変数への値代入指針

bool 変数名	説明
httpd_unified	対話式でユーザに入力を求める
httpd_builtin_scripting	LoadModule で php*_module が宣言されているかどうか
httpd_enable_cgi	ScriptAlias が宣言されているかどうか ExecCGI が宣言されているかどうか
httpd_enable_homedirs	UserDir ディレクティブが宣言されているかどうか
httpd_ssi_exec	Options +Includes が宣言されているかどうか
httpd_tty_comm	mod_ssl が用いられているかどうか 用いられているならば、秘密鍵に passpharse が必要かどうか
httpd_can_network_connect	LoadModule で mod_proxy が宣言されているかどうか

5.5.4 調整スクリプトの動作概要

まず、Apache の実行バイナリのパスを確定させる。調査スクリプトは Apache の実行バイナリである httpd, apache, apache2 のいずれかをあらかじめ設定されたディレクトリの中から検索を行う。同様に apachectl の検索を行う。Apache の実行バイナリ

及びコントロールスクリプトである apachectl が見つからなければスクリプトを終了させる。

次に、設定ファイルのパスを確定させる。”httpd -V”コマンドによって出力される情報は、コンパイル時に設定されたものである。”httpd -V”コマンドを実行すると、図5.6 の結果が得られる。有益な情報ではあるが、そのままでは実際のパスとして扱えない。その

```
-D SUEXEC_BIN="/usr/sbin/suexec"
-D HTTPD_ROOT="/etc/httpd"
-D SERVER_CONFIG_FILE="conf/httpd.conf"
```

図 5.6: ”httpd -V”コマンドの実行結果

ため、文字列処理を行ってシェル変数として取り込む。”*HTTP_ROOT*/SERVER_CONFIG_FILE”が、デフォルトの設定ファイルになる。ただし、Apache を起動するための rc スクリプトを調査し、-f オプションを用いて設定ファイルの再指定が行われていれば、再指定された設定ファイルを設定ファイルとして扱う。

また、”http -V”コマンドの結果、suexec が組み込まれていれば、取得されたパスに suexec があることを確認する。

以上がシステム構成及び、”httpd -V”コマンドから得られる情報である。

次に、httpd.conf を解析していく。httpd.conf は一行に一つのディレクティブと、ディレクティブに対して設定を行う value をスペース区切りで記述する。調整スクリプトは、文字列処理を行ってこの構文を解析する。ただし、”httpd -V”の場合と違う点がある。それは同じディレクティブが複数回に渡り登場し、また、それらの宣言は全て有効である点である。Security Context に付与する場合の利便性も考慮し、本ツールではあるディレクティブに設定された value を”:”で区切り、ディレクティブの名前で作られたシェル変数に代入する。解析の様子を図 5.7 に示す。

```
LoadModule access_module modules/mod_access.so
LoadModule auth_module modules/mod_auth.so
LoadModule auth_anon_module modules/mod_auth_anon.so

$LoadModule='access_module modules/mod_acess.so:\
             auth_module modules/mod_auth.so:\
             auth_anon_module modules/mod_auth_anon.so'
```

図 5.7: httpd.conf の解析

あるディレクティブに対して、設定された全ての値をまとめることで、第 5.5.2 節で述べた Security Context の付与が楽になる。相対パスは ServerRoot からの絶対パスに変更するなどの処理を通すと、上記の LoadModule の場合は図 5.8 のようになる。

```
@LOAD_MODULE@          --  system_u:object_r:httpd_module_t
     ↓このように置換される
((/etc/httpd/modules/mod_access.so)|(/etc/httpd/modules/mod_auth.so)\
|(/etc/httpds/modules/mod_auth_anon.so))\
                       --  system_u:object_r:httpd_module_t
```

図 5.8: 調整スクリプトによる置換例

この時、留意しなければならない問題がある。それは Symbolic Link の問題である。SELinux の Security Context は i-node に対応して付与されるものである。そのため、Symbolic Link とその Symbolic Link が導くファイルは別の Security Context である可能性がある。仮に ServerRoot が”/etc/httpd”とされている際に、”/etc/httpd/modules”という Symbolic Link を用意たとする。その参照先が”/usr/lib/modules/httpd”であれば、これらのモジュールの実体は”/usr/lib/modules/httpd/***.so”にあることが分かる。SELinux において、Symbolic Link を含むアクセスは Symbolic Link 自体と Symbolic Link の参照先へのアクセス権限が必要である。そこで、Symbolic Link が含まれていた場合には、Symbolick Link 自体とその参照先にラベルを付与する。

本節で述べたように調整スクリプトは動作する。

第6章　評価

本章では、本研究におけるフレームワークの正当性を検証するために、プロトタイプ実装によって自動生成されたセキュリティ・ポリシの評価を行う。

6.1　実験環境

表 6.1 に評価に用いた基本となるソフトウェア環境を示す。また、表 6.2 に評価に用いたハードウェア環境を示す。Fedora Core Linux4 に付属しているセキュリティ・ポリシは事実上の SELinux 標準であるため、デフォルトのセキュリティ・ポリシとする。また、このセキュリティ・ポリシは自動生成にベースとして用いたものもデフォルトのセキュリティ・ポリシとして用いたものも同様のバージョンを利用している。

表 6.1: 実験ソフトウェア環境

OS	Fedora Core Linux 4 (Kernel 2.6.14-1)
ソフトウェア	Apache 2.0.54 GNU bash 3.0.16 GNU sed 4.04
セキュリティ・ポリシ	strict-1.27.1-2.16

表 6.2: 実験ハードウェア環境

Machine	VMware GSX Server
CPU	Xeon 3.2GHz
Memory	1GB
HDD	4GB

6.2　正当性の評価

プロトタイプ実装によって自動生成されたセキュリティ・ポリシの正当性を評価する。

Apache を Fedora Core Linux4 パッケージのデフォルト設定に基づいて実行する。自動生成されたセキュリティ・ポリシを用いて正しく動作するか、具体的な評価項目に基づいて評価する。具体的な評価項目に対して、デフォルトの設定ファイルから推測されるアクセスの可否と自動生成されたセキュリティ・ポリシのアクセス制御を比較する。また、デフォルトのセキュリティ・ポリシとも比較を行う。評価項目と結果を表 6.3 に示す。

表 6.3: セキュリティ・ポリシの正当性の評価

評価項目	設定	自動生成	デフォルト
Apache の正常起動の可否	○	○	○
DocumentRoot 以下に置かれた html 等の表示可否	○	○	○
ScriptAlias 以下に置かれた cgi の利用可否	○	○	○
cgi から ScriptAlias 以下ファイルへの読込可否	−	○	○
cgi から ScriptAlias 以下ファイルへの書込可否	−	×	×
cgi から DocumentRoot 以下ファイルへの読込可否	−	×	×
cgi から DocumentRoot 以下ファイルへの書込可否	−	×	×
ScriptAlias 以下に置かれたファイルの表示可否	○	○	○
各種ログの出力の可否	○	○	○
ユーザディレクトリ以下の html 等の表示可否	×	×	○
ユーザディレクトリ以下の cgi 等の利用可否	×	×	×
cgi のネットワークアクセス利用の可否	−	×	×

まず、Apache の基本的な動作の可否を評価項目として取り上げる。Apache には様々な機能が備わっているが、今回は Apache の基本的な動作のみを取り上げる。例えば、正常に起動するかどうか、DocumentRoot 以下に置かれた html を正しく表示できるか、などである。設定の列は、それぞれの評価項目に対して設定ファイルから読み取った管理者の意図である。ScriptAlias が設定されており、ディレクトリが設定されていたとする。この場合、管理者はそのディレクトリで cgi を利用しようとしているといえる。このように、設定ファイルから管理者の意図を読み取ることができる。ただし、評価項目の内、cgi に関するアクセス制御の幾つかは設定ファイルから管理者の意図を読み取ることが難しい。本評価では、設定ファイルから管理者の意図を読み取ることができないものに関しては設定の項目を”−”とした。

評価の結果は表 6.3 のようになる。本研究により自動生成されたセキュリティ・ポリシは Apache を正常に起動できるかどうか、DocumentRoot 以下に置かれた html を正しく表示できるかどうかなど設定ファイルから読み取れる範囲で正しく生成することができた。ほとんどの項目でデフォルトのセキュリティ・ポリシと同様に動作した。動作が異なったのはユーザディレクトリ以下の html の表示可否に関する項目である。デフォルトの設定ファイルでは UserDir は disable とされている。このことから、ユーザ

ディレクトリ以下の html は公開しない意図を読み取ることができる。したがって、管理者の意図をより忠実に再現するためには、ユーザディレクトリ以下のファイルへの表示権限は不要である。本研究によって自動生成されたセキュリティ・ポリシはユーザディレクトリ以下のファイルに対して不要な権限を付与しなかった。自動生成したセキュリティ・ポリシは、デフォルトのポリシに比べ、ユーザディレクトリ以下の html の表示に関してより正確な定義を行うことができた。これは、自動生成が設定ファイルの UserDir ディレクティブに基づいて定義したためである。

6.3 セキュリティ強度の評価

セキュリティ強度の評価として、自動生成されたセキュリティ・ポリシを評価する。

管理者が望む動作をするのであれば、サービス・アプリケーション用に明示的に Security Context が付与されたリソースの数は少なければ少ないほどセキュリティ強度が高いといえる。なぜならば、Security Context を付与する行為は、すなわちアクセス権限を付与する定義であるからである。例えば、第 6.2 節の評価で述べたユーザディレクトリ以下のファイルの場合、Apache の設定ファイルで disable されている。そのため、正常なアクセスでは Apache はユーザディレクトリ以下へのファイルへアクセスを行わない。これらのファイルに不要なラベルが付与されていることは、攻撃者に Apache の権限を奪取された場合に問題がある。本来、Apache はアクセスするべきではないファイルに対して、アクセスを行うことができる可能性がある。このように、正常に動作する範囲で付与される権限は少なければ少ないほど良いことが分かる。

6.3.1 評価手順

Apache を Fedora Core Linux4 パッケージのデフォルト設定に基づいて利用する環境を想定する。

自動生成されたセキュリティ・ポリシによって実際に付与されたファイルの総数と、デフォルトの定義で実際に付与されたファイルの総数を比較する。

6.3.2 評価結果

結果を表 6.4 に示す。

自動生成ではパスなどの情報を実際に収集したものを用いる。そのため、あらかじめ多くの環境に適用するように定義されたセキュリティ・ポリシよりも正確に定義を行うことができた。

自動生成されたセキュリティ・ポリシと、デフォルトの定義との差異の原因について考察する。httpd_config_t タイプや httpd_modules_t タイプ、httpd_sys_content_t タイプが付与されたファイル数の差異は、ラベルの定義の仕方が原因である。デフォルトのセキュリティ・ポリシは個々のファイルに対してラベルを定義するのではなく、ディ

表 6.4: Security Context を付与されたファイル数

Security Context	自動生成	デフォルト
httpd_exec_t	1	1
httpd_suexec_exec_t	1	1
httpd_config_t	3	4
httpd_sys_content_t	248	249
httpd_sys_script_exec_t	2	2
httpd_user_content_t	0	7
httpd_log_t	3	3
httpd_modules_t	43	48
httpd_var_run_t	1	1
合計	302	316

レクトリと正規表現を用いてラベルの定義を行う。一方で、自動生成されたセキュリティ・ポリシは、設定ファイルに基づき、実際に利用する個々のファイルに対してラベルの定義を行った。この差がラベルを付与されたファイルの差になったと考えられる。

例として httpd_sys_content_t タイプの Security Content の定義を図 6.1 に示す。Fedora Core Linux パッケージのデフォルト設定における DocumentRoot は/var/www/html である。しかし、Alias ディレクティブを用いて/var/www/error や/var/www/icons に DocumentRoot と同様の静的なコンテンツを置いている。これに対して、デフォルトのセキュリティ・ポリシは/var/www(/.*)?に Security Context の定義を行うことで対処している。また、/var/www 以下が定義されているにも関わらず、同様に DocumentRoot を置くのに一般的な/srv/([^/]*/)?www/(/.*)?が定義されている。これは、様々な環境に対応しようとした結果である。したがって、不要な定義であり、これらの定義をしない場合に比べて脆弱であるといえる。

```
% 自動生成されたセキュリティ・ポリシ
/var/www/html(/.*)?             system_u:object_r:httpd_sys_content_t
((/var/www/error)|(/var/www/icons))(/.*)?
                                system_u:object_r:httpd_sys_content_t

% デフォルトのセキュリティ・ポリシ
/var/www(/.*)?                  system_u:object_r:httpd_sys_content_t
/srv/([^/]*/)?www(/.*)?         system_u:object_r:httpd_sys_content_t
/var/www/icons(/.*)?            system_u:object_r:httpd_sys_content_t
```

図 6.1: httpd_sys_content_t の定義

また、実験を行った環境では、ユーザディレクトリ以下での html の公開を許していない。自動生成されたセキュリティ・ポリシは設定ファイルから得られた情報を用いて、ユーザディレクトリ以下にラベルを付与することはなかった。しかし、デフォルトのセキュリティ・ポリシには多くの環境に適用するようにあらかじめ定義されていたため、不必要なラベルの付与を行ったと考えられる。

6.4 定義工数の評価

定義工数の評価として、セキュリティ・ポリシを定義するために掛かる時間の計測を行った。

6.4.1 評価手順

Apache を apache.org からダウンロードしたソースからコンパイルを行い、apache.org 標準の環境と設定を用いる環境を想定する。

本研究を用いてセキュリティ・ポリシの自動生成を行うのに要した時間を計測する。計測は 150 回行い、その平均を用いる。比較対象として、デフォルトのセキュリティ・ポリシの変更を手動で行う場合と、セキュリティ・ポリシを一から定義した場合に必要な行数を計測した。手動で定義を行った場合の時間は必要な行数× 15 秒で計算するものとする。

手動で変更を行う場合は自動生成と同じセキュリティ・ポリシになるように変更を行う。一から定義を行う場合は、実際に一から定義を行うと、最終的に同様のセキュリティ・ポリシが生成される保証がない。そのため、自動生成されたセキュリティ・ポリシ全体の行数を計測した。これを一から定義を行ったものだと仮定する。

6.4.2 評価結果

結果を表 6.5 に示す。

表 6.5: セキュリティ・ポリシの定義に必要な工数

手法	行数	秒数	標準偏差
自動生成	—	157.453	7.269
手動変更	73	1095	—
一から定義	376	5640	—

表 6.5 の通り、自動生成を導入することで、セキュリティ・ポリシの定義コストを削減することができた。

自動生成を行う場合には、以下のような手順を踏むことになる。本研究を利用するユーザは自動生成を行うためにスクリプトを実行するだけで良い。また、ただ実行するだけではリソースのパスの検索に失敗する場合には、パラメータを指定する必要がある。調整スクリプトはベースとなるデフォルトのセキュリティ・ポリシをよく理解しているものによって作られる。ユーザはあるラベルに付与されているアクセス権限が何であるかを意識する必要がない。そのため、セキュリティ・ポリシを意識することなく利用が可能である。

- 調整スクリプトを実行する
- リソースの検索に失敗した場合、パラメータでリソースを指定する

手動で変更を行う場合には、一行辺り 15 秒という仮定のもと、秒数を計算したが、実際には以下のような手順を踏まなければならない。まず、変更を行う対象のセキュリティ・ポリシに関する理解が必要である。デフォルトのセキュリティ・ポリシに設定されている内容を理解しなければ、変更する箇所を特定することができない。しかし、SELinux のセキュリティ・ポリシは難解であり容易には理解できない。また、再定義と制御の選択を行った後も、変更を行ったセキュリティ・ポリシが正しく動作するかを確認しなければならない。このように、本評価では一行 15 秒としたが、実際にはもっと多くの時間がかかる。

- デフォルトのセキュリティ・ポリシの理解
- ラベルの再定義と適用するアクセス制御の選択
- 変更したセキュリティ・ポリシの正当性チェック

第7章　結論

7.1　まとめ

セキュアOSは従来のOSには存在しなかった強制アクセス制御と最小特権を実現する。これにより、プログラムの脆弱性を狙った攻撃に対抗することができる。しかし一方、厳格なアクセス制御を行うために必要な定義が複雑である。

セキュアOSの一つにSELinuxがある。SELinuxも他のセキュアOSと同様に、セキュリティ・ポリシと呼ばれるアクセス制御の定義が必要である。SELinuxは特に非常に粒度の細かいアクセス制御を行うことができる。しかし、非常に粒度の細かいアクセス制御を行う性質上、セキュリティ・ポリシは複雑である。そのため、SELinuxの運用・導入コストは非常に大きい。

SELinuxの運用・導入コストを削減するために幾つかの先行研究[15][17]が存在する。しかし、いずれもその利用にセキュリティ・ポリシに関する知識を必要とする。そのため、SELinuxの運用・導入コストを大幅に削減できたとはいえない。

本研究は、セキュリティ・ポリシの自動生成を提案をする。セキュリティ・ポリシに精通している者があらかじめ定義したセキュリティ・ポリシをベースとして自動調整を行う。調整はセキュリティ・ポリシを生成する対象のサービス・アプリケーションの設定ファイルとシステムの構成に基づいて行う。これにより、セキュリティ・ポリシの知識を必要としない自動生成が可能になる。また、自動生成を実現するためのフレームワークの提案と設計を行った。セキュリティ・ポリシに精通している人間の知識を共有・流通させる。これにより、SELinuxの導入・運用コストの削減が可能である。SELinuxやセキュリティ・ポリシの知識に精通していなくても利用が可能となる。

プロトタイプとしてApacheの自動生成ツールの実装を行った。実装はフレームワークに準じた。これにより、ホストに合わせたApacheのセキュリティ・ポリシの自動生成を実現した。

幾つかの項目を設けてApacheの動作確認を行い、自動生成されたセキュリティ・ポリシの正当性を確認した。セキュリティ強度を不必要な権限の付与が行われた数によって評価した。デフォルトのセキュリティ・ポリシより不必要な権限の付与を削減できた。また、セキュリティ・ポリシを定義する工数について評価を行った。一からセキュリティ・ポリシを定義した場合、デフォルトのセキュリティ・ポリシに手動で変更を加えた場合と比較して工数を大幅に削減できた。

本研究の成果により、SELinuxやセキュリティ・ポリシの知識に精通していなくても利用が容易となり、SELinuxの普及が期待できる。

7.2 今後の課題

今後の課題として、調整スクリプトの記述のコストの削減が求められる。本フレームワークはユーザのコストを大幅に削減する。しかし一方、自動生成を行うためのスクリプトを定義する者に要求されるコストが非常に大きい。本研究と同様の試みに configure がある。configure スクリプトの記述は autoconf[25] を用いて簡易化されている。そのため、ソフトウェアの開発者が容易に記述できる。本フレームワークにおいても、マクロなどによる記述の容易化が課題となる。

本研究ではプロトタイプとして Apache の自動生成を行った。Apache の場合は設定ファイルからセキュリティ・ポリシを変更する情報を取得できた。しかし、同様のことが全てのサービス・アプリケーションで行えるかは不明である。Apache だけではなく、他の多くのサービス・アプリケーションで同様に自動生成が行えるかどうかの確認が課題となる。

また、設定ファイルなどから得られる情報では、付与するべきラベルが判断出来ない場合がある。例として、Apache における cgi 関係のアクセス権限がある。cgi スクリプトがアクセスすべき対象や、その対象に対してどのようなアクセスを許可すべきかを判断することは難しい。デフォルトのセキュリティ・ポリシにおいても、cgi に関するファイルのアクセス権限は、ユーザが任意に再定義を行うことを解決策としている。再定義には chcon コマンド（第 2.2.1 節参照）を用いる。chcon コマンドの利用には、セキュリティ・ポリシに関する知識が必要とされる。このように、定義を行う段階で正しいラベルが判断できない場合がある。したがって、ラベルの付与の簡易化にも取り組む必要がある。

7.3 今後の展開

一般利用者にデフォルトのセキュリティ・ポリシをバイナリのみで提供しようという動向があることは第 2.4 節で述べた。本フレームワークは、アクセス制御の定義の選択には Conditional を利用している。また、ラベルの付与の定義はバイナリではなくテキストのまま提供される。そのため、本研究の成果はバイナリで提供されるポリシにも適用可能である。

また、付録に掲載した Loadable Policy Modules がある。Loadable Policy Modules はセキュリティ・ポリシをパッケージ管理するためのシステムである。これにより、セキュリティ・ポリシを容易にサービス・アプリケーション毎に分けて管理・導入することができる。Loadable Policy Modules もバイナリの形で提供される。本研究はポリシ・バイナリであっても適用可能である。したがって、Loadable Policy Modules にも適用可能である。Loadable Policy Modules は、checkmodule コマンドを用いてセキュリティ・ポリシの導入を行う。このコマンドに本研究の成果を組み込むなどの展開が考えられる。

謝辞

本論文の作成にあたり、御指導いただきました慶應義塾大学環境情報学部教授村井純博士、並びに同学部教授徳田英幸博士、同学部助教授楠本博之博士、同学部助教授中村修博士、同学部助教授高汐一紀博士、同学部専任講師湧川隆次博士に感謝いたします。

常日頃からお世話になりました慶応義塾大学院政策・メディア研究科講師南政樹氏、同研究科博士課程小原泰弘氏、同研究科修士課程白畑真氏に深く感謝したします。

また、論文作成にあたり、多大なご助力とご助言をいただきました、日立ソフトウェアエンジニアリング株式会社技師中村雄一氏に深く感謝いたします。

本論文作成にあたり、ご支援をいただきました慶應義塾大学環境情報学部金井瑛氏、同学部奥村佑介氏、並びに慶應義塾大学徳田・村井・楠本・中村・高汐・湧川合同研究室の皆様、特に SING/IA*研究グループの皆様に感謝いたします。

参考文献

[1] セキュアOSと基盤ソフトウエアに関する研究会. セキュアOSと基盤ソフトウエアに関する研究会. http://secure-os.yosihiro.com/.

[2] National Security Agency. Security-Enhanced Linux. http://www.nsa.gov/selinux/.

[3] LIDS Project. Linux Intrusion Detection System. http://www.lids.org/.

[4] Trusted BSD Project. Trusted BSD. http://www.trustedbsd.org/.

[5] Sun Microsystems. Trusted Solaris. http://jp.sun.com/products/software/solaris/trustedsolaris/.

[6] Infocom Corporation. Pitbull. http://www.pitbull.jp/.

[7] Peter Loscocco, National Security Agency Stephen Smalley, NAI Labs. Integrating Flexible Support for Security Policies into the Linux Operating System. *http://www.nsa.gov/selinux/papers/freenix01.pdf*.

[8] Mozilla Corporation. Thunderbird – Reclaim your inbox. http://www.mozilla.com/thunderbird/.

[9] CERT. CERT Advisory CA-2002-27 Apache/mod_ssl Worm. http://www.cert.org/advisories/CA-2002-27.html.

[10] The Apache Software Foundation. The Apache HTTP Server Project. http://httpd.apache.org/.

[11] OpenSSL Project. OpenSSL. http://www.openssl.org/.

[12] GNU Project - Free Software Foundation. GNU m4. http://www.gnu.org/software/m4/.

[13] Fedora Core Project. Fedora Core Linux. http://fedora.redhat.com/.

[14] Tresys Technology. SEFramework - Symposium. http://www.selinux-symposium.org/2005/presentations/session6/6-1-wilson.pdf, 2005.

[15] The MITRE Corporation. Automated Policy Generation. http://www.mitre.org/tech/selinux/.

[16] Daniel H. Jones. Virgil: SELinux Policy Generator. http://sepolicy-virgil.sourceforge.net/.

[17] 中村 雄一, 鮫島 吉喜. Security-Enhanced Linux のアクセス制御ポリシ設定の簡易化. *Symposium on Cyrptography and Information Security 2003*, 2003.

[18] SELinux Policy Editor Project. SELinux Policy Editor Project. http://seedit.sourceforge.net/.

[19] 田端 利宏，末安 克也，櫻井 幸一. 設定ツールによる SELinux アクセスパーミッション統合の安全性評価. **情報処理学会論文誌 テクニカルノート，***Vol.46***，***No.4*, pages 1070–1073, Apr 2005.

[20] Red Hat. Red Hat. http://www.redhat.com/.

[21] Tresys Technology. Tresys Technology. http://www.tresys.com/index.html.

[22] Yuuichi Nakamura. Meaning of permissions in SELinux(Ver 1). http://seedit.sourceforge.net/doc/access_vectors.pdf, Jan 2005.

[23] . The Webalizer. http://www.mrunix.net/webalizer/.

[24] Tresys Technology. SELinux Reference Policy Project. http://www.tresys.com/tech/refpol.shtml.

[25] GNU Project, Free Software Foundation. Autoconf -GNU Project- Free Software Foundation (FSF). http://www.gnu.org/software/autoconf/.

第8章　付録

8.1　Loadable Policy Modules

Loadable Policy Modules は Tresys が中心となって開発を進めている SELinux Policy Management Framework の一つである。セキュリティ・ポリシをモジュール・パッケージとして管理する。Fedora Core 5 test-1 に採用されている。セキュリティ・ポリシの定義の際に Loadable Policy Modules 独自の定義が必要になる。Loadable Policy Modules 独自の定義を図 8.1 に示す。

```
module syslogd 1.0
require { attribute domain; }
optional { require { type klogd_t; }
       allow klogd_t domain : process signal; }
```

図 8.1: Loadable Policy Modules

module

module の名前と Version の定義を行う。module はこの定義によって管理される。

require

module を導入するにあたって必要なセキュリティ・ポリシの定義を定義する。この場合であれば、domain という属性が定義されていなければ、この module を導入することはできない。

optional

require は必要不可欠な依存性の解決であるが、optional は依存性の任意解決を行う。optional 構文内の require に定義された要素が定義されていた場合に展開される。図 8.1 であれば klogd_t の定義が存在していた場合に、展開される。これは m4 における ifdef に該当するものである。

Loadable Policy Modules を導入することで、ポリシをモジュールで管理できるようになるため、ポリシのパッケージ管理が容易になる。また、Loadable Policy Modules は適用されるポリシの形態を選ばない。

ポリシ・モジュールのパッケージは、モジュールと呼ばれるセキュリティ・ポリシ・バイナリと file_contexts から構成されている。そのため、ポリシ・モジュールを組み込むだけで、あるサービス・アプリケーションに関するセキュリティ・ポリシを導入することができる。

8.2 Reference Policy

Reference Policy は依存性などのポリシ管理の問題を解決するために作られた新しいセキュリティ・ポリシのフォーマットである。Reference Policy もまた Tresys を中心として開発が進められている。

従来のフォーマットのポリシは主に file_contexts の定義を行う.fc ファイルと主に TE に関わる定義を行う.te ファイルに分かれている。サービス・アプリケーション毎にファイルを分けられていたが、依存性の解決に関しては明確なルールは存在しなかった。また、第 2.3.1 節で述べた audit2allow を用いて作られたポリシが多く、マクロにも明確なルールが存在しなかった。Reference Policy は現状のセキュリティ・ポリシを整理して開発されている。.fc ファイル、.te ファイルに加え、インタフェース用のマクロを定義した.if ファイルに分け管理を行う。audit2allow によって作られたポリシは一から手で書いたポリシに比べて脆弱である。Reference Policy は整理されたマクロを用いて記述する。これにより、一から手でポリシを書くことが容易になる。また、依存性の解決もマクロを用いて行うことで容易になる。しかし、一方でマクロの数が膨大になっている。SELinux に精通していないユーザが Reference Policy を用いて一から手でポリシを書くことは現実的に不可能である。

8.3 ポリシ・テンプレート

プロトタイプ実装で作成したポリシ・テンプレートの apache.fc.in を図 8.2 に示す。apache.te.in はポリシ・テンプレートのベースとした Strict ポリシの apache.te をそのまま利用できる。

```
# binary as follows
@APACHE_BIN@              --    system_u:object_r:httpd_exec_t
@APACHECTL_BIN@           --    system_u:object_r:initrc_exec_t
@SUEXEC_BIN@              --    system_u:object_r:httpd_suexec_exec_t

# config as follows
@SERVER_ROOT@             -d    system_u:object_r:httpd_config_t
@HTTPD_CONF@              --    system_u:object_r:httpd_config_t
@INCLUDE@                       system_u:object_r:httpd_config_t

# logs as follows
#@LOG_DIR@                -d    system_u:object_r:httpd_log_t
@SERVER_ROOT@/logs              system_u:object_r:httpd_log_t
@ERROR_LOG@               --    system_u:object_r:httpd_log_t
@COOKIE_LOG@              --    system_u:object_r:httpd_log_t
@CUSTOM_LOG@              --    system_u:object_r:httpd_log_t
@TRANSFER_LOG@            --    system_u:object_r:httpd_log_t
@REWRITE_LOG@             --    system_u:object_r:httpd_log_t
@SCRIPT_LOG@              --    system_u:object_r:httpd_log_t

# modules as follows
#@MODULE_DIR              -d    system_u:object_r:httpd_modules_t
@SERVER_ROOT@/modules           system_u:object_r:httpd_modules_t
@LOAD_MODULE@             --    system_u:object_r:httpd_modules_t

# misc as follows
@PID_FILE@                --    system_u:object_r:httpd_var_run_t
@LOCK_FILE@               --    system_u:object_r:httpd_var_run_t
@SCORE_BOARD_FILE@        --    system_u:object_r:httpd_var_run_t

# content as follows
@DOCUMENT_ROOT@(/.*)?           system_u:object_r:httpd_sys_content_t
@ALIAS@(/.*)?                   system_u:object_r:httpd_sys_content_t
@SCRIPT_ALIAS@(/.*)?            system_u:object_r:httpd_sys_script_t
@USER_DIR@(/.+)?                system_u:object_r:httpd_ROLE_content_t
```

図 8.2: apache.fc.in