2007－CSEC－39 (5)
2007／12／14

インターネットサーバにおけるセキュアOSのポリシー記述簡略化

横山 敏博　花岡 美幸　嶋村 誠　河野 健二

慶應義塾大学 理工学部 情報工学科

E-mail: {burger, hanayuki, tima}@sslab.ics.keio.ac.jp, kono@ics.keio.ac.jp

近年，インターネットサーバに対する不正アクセスが問題となっており，その対策としてセキュアOSが注目されている．しかし，セキュアOSはポリシー記述が煩雑であり，設定に多大な労力を要する．本論文では，外部との接続が確立した後でしかリモート攻撃は起こりえない点に着目し，実行時のフェーズを考慮したポリシー記述簡略化を提案する．フェーズを考慮したポリシー記述では，システムが起動してから外部と接続が確立するまでの初期化フェーズと，接続が確立した後のプロトコル処理フェーズとを考え，初期化フェーズではアクセス制御を行わず，プロトコル処理フェーズでのみアクセス制御を行う．結果として，サーバがサービスを開始するまでの前処理に必要なポリシーを削減することができる．実際に，HTTP，SMTP，POPの三種類のサーバに対してポリシーを記述し，簡略化できることを確認した．

Simplifying Security Policy Descriptions for Internet Servers in Secure Operating Systems

Toshihiro Yokoyama　Miyuki Hanaoka　Makoto Shimamura　Kenji Kono

Department of Information and Computer Science, Keio University
E-mail: {burger, hanayuki, tima}@sslab.ics.keio.ac.jp, kono@ics.keio.ac.jp

In today's Internet, secureOSes are widely used to prevent unauthorized access to servers. In secure OSes, however, describing security policies properly is hard and quite a challenge for administrators. Considering that remote attackers can never attack against servers which do not establish connections to them, this paper shows a scheme that exploits phases in execution to simplify security policy descriptions. We define an initalization phase before servers establish connections to their clients and a protocol processing phase after servers complete establishing connections. Access control is enforced only in a protocol processing phase. Therefore, we can eliminate a policy description until servers estabish connections. We describe security policies for three kinds of Internet servers (HTTP, SMTP, POP) and demonstrate that our scheme can simplify security policy descriptions.

1 はじめに

インターネットサーバへの不正アクセス対策として，セキュアオペレーティングシステム (以下，セキュアOS) を用いることが増えている．セキュアOSとは一般的なオペレーティングシステム (以下，OS) と比べて，アクセス制御を強化したOSであり，細粒度のアクセス制御を提供する．セキュアOSでは，管理者権限が存在せず，プロセス単位でアクセス制御を行う仕様になっているため，たとえ不正アクセスが成功してサーバの制御を乗っ取られたとしても，攻撃者は乗っ取りに成功したプロセスに許可されている限られた操作しかできず，システム全体の壊滅のような深刻な被害を防ぐことができる．

しかし，セキュアOSはセキュリティポリシーの設定が煩雑であるため，導入・保守のコストが高い．セキュリティポリシーとは，各プロセスに対してアクセス可能なリソースを定めたものである．セキュアOSでは，プロセスはポリシーによって許可されたリソースのみへアクセスすることができ，許可されていないリソースへのアクセスはできない．したがって，セキュアOSを導入する場合，管理者は各プロセスが必要とするリソースを把握し，それらを許可するポリシーを全て正確に記述しなければならない．また，サーバのバージョンアップ等によって必要とするリソースが変わった場合，管理者はその度にポリシーを再設定する必要がある．これらは管理者に相当な手間とシステムに関する専門的な知識を要求する．

本論文では，インターネットサーバを対象としたセキュアOSのポリシー記述簡略化手法を提案する．外部からのリモート攻撃を防御するという用途に限定した場合，通信相手との接続が確立がするまではサーバが攻撃を受けることはない．したがって，通信相手との接続が確立した時からセキュアOSによるアクセス制御を適用すればよく，サーバがサービスを開始するまでの前処理に必要なポリシーの記述を削減できる．セキュアOSでは，最小権限の原則[9]に従ってシステムの起動時からの全プロセスに対して必要最小限の権限のみを与える．しかし，この

ような前処理のためのポリシー設定は非常に煩雑である．なぜなら，前処理ではシステムの起動，ファイルシステムのマウント，ネットワークの設定等の様々な処理が行われ，サーバが起動してからサービスを開始するまでにも設定ファイルの読み込みやライブラリのリンク等の処理を行う．リモート攻撃対策としてセキュアOSを導入する場合，前処理のためのポリシー記述を簡略化することができ，記述の簡略化が見込める．

そこで本論文では，通信相手との接続が確立した後でなければリモート攻撃は起こりえない点に着目し，**実行時のフェーズ**を考慮したポリシー簡略化を提案する．実行時のフェーズとは，システムの処理の流れを，システム外部からメッセージを受け取る前までの**初期化フェーズ**とメッセージを受け取った後の**プロトコル処理フェーズ**に分けて考えることである．初期化フェーズでは，リモート攻撃を受けることがないのでアクセス制御を行わず，プロトコル処理フェーズでのみアクセス制御を行う．結果として，初期化フェーズでアクセスするリソースに対するポリシー記述が不要なため，ポリシー記述量を削減できる．また，初期化フェーズで行われる前処理についての専門的知識が不要になり，記述難易度の低下につながる．

実際に，セキュアOSの一つであるSELinux[1]に提案機構を実装し，HTTP，SMTP，POPの三種類のインターネットサーバのポリシー記述を行い，その有効性の検証を行った．その結果，前処理のためのポリシー記述を行ったポリシーに比べて，それぞれ59.3%，14.6%，23.5%ルール数を削減できた．

2 ポリシー記述の煩雑さの分析

2.1 セキュアOSにおけるポリシー

セキュアOS上で，各プロセスに対してアクセス可能なリソースを定めたものを，セキュリティポリシーと呼ぶ．セキュアOSにおけるポリシーは，最小権限の原則に従って各プロセスの実行に必要最小限のアクセス許可のみを記述する．

代表的なセキュアOSであるSELinuxでは，Type Enforcement (TE)[2]と呼ばれるアクセス制御モデルを用いており，サブジェクト，オブジェクトに**ドメイン**，**タイプ**と呼ばれるラベルを付与し，**アクセスベクタ**と呼ばれるパーミッションを与えることにより，アクセス制御を行う．オブジェクトについては，

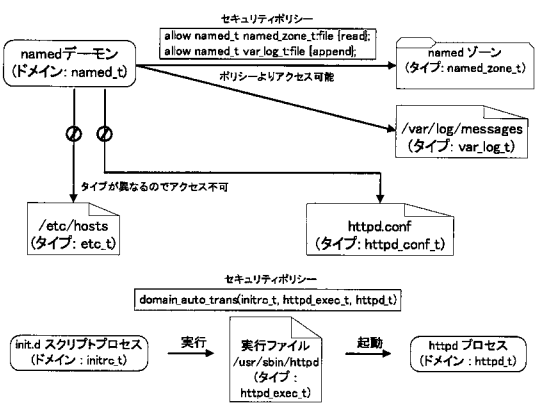

図1: TEによるアクセス制御

ファイルやディレクトリ等の**オブジェクトクラス**を定めている．

TEによるアクセス制御の例を図1に示す．この図は，namedデーモンがサービスを行うのに必要とするnamedゾーン関連のファイルの読み込み，ログファイルへの追記を許可し，その他のリソースへのアクセスは禁止するといった例である．

```
allow named_t named_zone_t:file {read};
```

というポリシーにより，named_tドメインを付与されたプロセス(この例では，namedデーモン)がnamed_zone_tタイプを付与されたfileオブジェクトクラス(この例では，namedゾーン関連のファイル)を読む込むためのreadアクセスベクタを与えている．namedデーモンはポリシーで許可されていない/etc/hostsやhttpd.confへのアクセスは一切行うことができない．

また，プロセスがファイルを実行するためにforkした場合，新たに生まれた子プロセスは通常では親プロセスと同じドメインで動作する．しかし，ポリシーで指定することにより，ファイルを実行した際に親プロセスとは別のドメインで子プロセスを動作させることができ，これを**ドメイン遷移**と呼ぶ．例えば，

```
domain_auto_trans
(initrc_t, httpd_exec_t, httpd_t)
```

というポリシーにより，initrc_tドメインを付与されたプロセス(この例では，init.dスクリプトプロセス)がhttpd_exec_tタイプを付与されたプログラム(この例では，/usr/sbin/httpd)を実行すると，そのプログラムはhttpd_tという新たなドメインで動作する．

2.2 ポリシー記述の煩雑さ

適切なポリシーを定めるには，多くのプロセスに対して，必要とするリソースへのアクセスを許可するポリシーを記述しなければならない．さらに，最小権限の原則に従ってポリシーを記述するには，システムに関する専門的知識が必要となる．

システムの起動時から様々なプロセスが生まれ，多様なリソースにアクセスする．特に各プロセスの初期化の段階では，多くのディレクトリに散在する設定ファイルを読みにいったり，OS 等の実行環境に依存するシステムコールを発行したりと非常に複雑な動きをする．例えば，Web サーバの Apache では，初期化処理において mod_access.so や mod_auth.so 等のモジュールの組み込み，libexpat.so や libkrb.so, libcurl.so 等のライブラリのリンク，httpd.conf や php.conf, ssl.conf 等の設定ファイルの読み込みなどを行う．

```
<src-phase>初期化フェーズ</src-phase>
<dst-phase>
        HTTP 処理フェーズ
</dst-phase>
<condition>
    <event domain="httpd_t">
        accept
    </event>
</condition>
```

図 2: Apache のフェーズ遷移記述例

3 ポリシー記述の簡略化

3.1 フェーズを考慮したポリシー記述

インターネットを介した外部からの攻撃を想定した場合，インターネットサーバが外部との通信を確立するまでは，サーバはいわば“きれいな”(untainted) 状態であり，攻撃を受ける可能性はない．したがって，外部と通信を確立するまでは，セキュア OS によるアクセス制御を行う必要がない．しかし，ひとたび外部との通信が確立し，通信メッセージを読み取ると，サーバは“汚染”(tainted) されている可能性がある．つまり，サーバは外部からの攻撃を受ける可能性があるので，アクセス制御を行う必要がある．

フェーズを考慮したポリシー記述では，システムが起動してから外部との接続が確立するまでの**初期化フェーズ**と，接続が確立した後の**プロトコル処理フェーズ**とを考え，初期化フェーズではアクセス制御は行わず，プロトコル処理フェーズでのみアクセス制御を行う．

初期化フェーズでは，2.2 節で述べたような様々な前処理を行う．それに対し，プロトコル処理フェーズでは，サーバ本来の処理だけが行われるため，アクセスするファイルや実行するシステムコールが比較的想定しやすくなる．

以上より，提案手法では，初期化フェーズでアクセスするリソースに対するポリシー記述が不要となりポリシー記述量を削減できる．また，初期化フェーズで行われるサーバがサービスを提供するまでの前処理に対する専門的知識が不要になり，記述難易度の低下につながる．

フェーズを考慮したポリシー記述では，初期化フェーズからプロトコル処理フェーズへと遷移するタイミングを決める必要がある．現在の実装では，初期化フェーズからプロトコル処理フェーズへの遷移は，ネットワーク接続の要求を行う connect システムコールとネットワーク接続の確立を行う accept システムコールを契機としている．ネットワーク通信が確立した場合，その後はサーバが“汚染”されている可能性があるため，初期化フェーズからプロトコル処理フェーズへと遷移し，セキュア OS によるアクセス制御を適用する．

フェーズを考慮したポリシー記述では，各フェーズごとにセキュア OS のポリシーを用意する．初期化フェーズでは，アクセス制御を行う必要がないため，全てのドメインに全ての行動を許可するポリシーを記述する．プロトコル処理フェーズでは，外部とのやりとりを行うドメインに対して最小権限を与え，その他の乗っ取られる可能性がないドメインに対しては全ての行動を許可するポリシーを記述する．

また，フェーズの遷移を行うため，フェーズの遷移条件を指定するための設定ファイルを XML を用いて作成した．例えば，Web サーバの Apache が accept を実行したときに初期化フェーズからプロトコル処理フェーズへと遷移するように指定するには，フェーズ遷移条件を図 2 のように記述する．httpd_t は Apache に付与されているドメインであり，accept の実行を契機としたフェーズの遷移に伴い，セキュア OS のセキュリティポリシーを初期化フェーズのポリシーから HTTP 処理フェーズのポリシーへと切り替えることにより，アクセス制御を提供する．

フェーズの遷移は基本的には初期化フェーズから

プロトコル処理フェーズへの一度しか行われない．しかし，SMTP サーバと POP サーバ等二つ以上のサーバを 1 つの計算機で動かしている場合は，初期化フェーズから SMTP 処理フェーズまたは POP 処理フェーズへ遷移した後に，SMTP プロトコルと POP プロトコルの両方を扱う SMTP-POP 処理フェーズへ再び遷移することがある．

3.2 セキュリティ強度とポリシー記述量

セキュリティ強度とポリシーの記述量にはトレードオフの関係がある．提案手法ではポリシー記述を外部から攻撃を受ける可能性が生じた後に限定できる代わりに，内部犯行による情報漏洩等は防ぐことができない．

これは targeted ポリシー [5] にも同様のことが言える．targeted ポリシーは SELinux で利用できるポリシータイプの 1 つであり，一部のデーモン (httpd, dhcpd, mailman 等) に対して最小権限の原則に従ったアクセス制御を適用し，その他のサブジェクトにはアクセス制御が適用されない unconfined_t ドメインを適用する．したがって，unconfined_t で動作しているプロセスを利用して，内部犯行が行われる可能性がある．

それに対して，もう 1 つのポリシータイプである strict ポリシー [4] では，システムの起動時から全てのプロセスがセキュア OS によるアクセス制御を適用されたドメインで動作する．したがって，内部犯行にも対応することができるセキュリティ強度を持つ．しかし，strict ポリシーは，システムの起動時から全プロセスに対して最小権限の原則に従ったポリシー記述が必要なため，targeted ポリシーに比べて設定が非常に煩雑である．

提案手法のポリシーは targeted ポリシーに近い．しかし，targeted ポリシーではポリシーを動的に切り替えることができないため，アクセス制御の対象である全てのデーモンに対して初期化処理のためのポリシー記述を行わなければならない．提案手法のポリシーは，targeted ポリシーと同様のセキュリティを備えた上で，targeted ポリシーよりもさらなるポリシー記述の簡略化が見込める．

4 実装

提案手法の有効性を検証するため，セキュア OS として SELinux，カーネルに Linux カーネル 2.6.18 を用いて，Fedora Core 4 上に提案手法を実装した．図 3 に提案機構のアーキテクチャを示す．

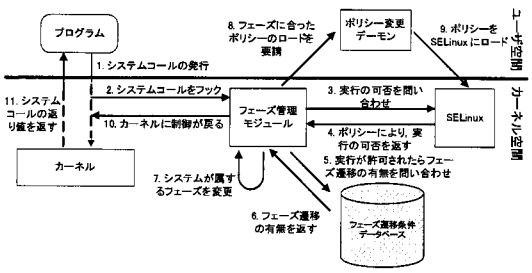

図 3: 提案機構のアーキテクチャ

フェーズ管理モジュールは，フェーズ遷移を検出するためにシステムコールのフックを行い，システムが属するフェーズの変更，ポリシー変更デーモンへ新たなポリシーのロードを要請するモジュールである．また，モジュールのロード時に，前もって記述されたフェーズ遷移条件をユーザ空間より受け取り，フェーズ遷移条件データベースをメモリ上に作成する．

ポリシー変更デーモンは，フェーズ管理モジュールからの命令を受けて，指定されたポリシーへと SELinux のポリシーを変更するためのデーモンである．SELinux ではポリシーはユーザ空間から変更されるように設計されており，実装の簡略化のため，SELinux で利用されているポリシーの変更を行うための load_policy プログラムを利用して作成した．

connect，accept システムコールが発行されると，フェーズ管理モジュールはそれらをフックし，フェーズ遷移の必要があるかどうかを調べる．必要がある場合，新たなフェーズのセキュリティポリシーのロードをポリシー変更デーモンへと要求し．ポリシー変更デーモンは指示されたポリシーを SELinux にロードする．

5 実験

提案手法の有効性を確認するため，HTTP，SMTP，POP サーバを対象として，ポリシー記述簡略化の評価，フェーズ遷移のオーバーヘッドの測定を行った．HTTP サーバは Apache 2.0.54，SMTP サーバは sendmail 8.13.8，POP サーバは dovecot 0.99.14 である．

5.1 ポリシー記述の簡略化

提案手法のポリシーとシステムの起動時からアクセス制御を行うポリシーとの比較を行うため，実際にシステム起動時からアクセス制御を行うポリシーを記述した．ドメインとタイプのラベル付けは，Fedora Core 4 の SELinux で標準にインスト-

ルされている strict ポリシーを利用した．また，初期化で動作させるデーモンはインターネットサーバにおいて必要と思われるものを選択し，具体的には acpid, anacron, apmd, atd, auditd, autofs, bluetooth, canna, cpuspeed, crond, haldeamon, iiim, iptables, kudzu, mDNSResponder, mdmonitor, messagebus, netfs, network, nifd, pcmcia, rhnsd, sshd, syslog とした．

システム起動時からアクセス制御を行うための strict ポリシーを記述した結果，サーバプログラムを実行するまでに必要なポリシーのルール数は 1438 であった．

また，提案手法を用いて，HTTP，SMTP，POP サーバに対してポリシーを記述した．HTTP は一つの HTML ファイルを GET するためのポリシー，SMTP ではメールを送信，リレーするためのポリシー，POP はユーザ認証を行い，メールを受信するためのポリシーを記述した．

フェーズの遷移条件は，Apache, sendmail では，初めて外部との通信を確立するまでを初期化フェーズ，その後をそれぞれ HTTP 処理フェーズ，SMTP 処理フェーズと定義し，サーバが発行する accept を契機としてフェーズ遷移を行うように定義した．dovecot では，ユーザ認証用のプロセスと実際のトランザクションを処理するプロセスが別プロセスとして動作する．ユーザがサーバにアクセスすると，始めにユーザ認証用プロセスがユーザとの通信を行うため，ユーザ認証用プロセスが初めてユーザとの通信を確立するまでを初期化フェーズ，その後を POP 処理フェーズと定義し，認証用プロセスが発行する accept を契機としてフェーズ遷移を行うように定義した．

サーバの初期化のためのポリシー記述が必要な strict ポリシーと，提案手法のポリシーとの比較を表 1 に示す．個々の数字はポリシーのルール数を表す．ここで一つのルールとは，ドメインに対してある一つのタイプの一つのオブジェクトクラスのためのアクセスベクタを定義することである．

表 1 より，全てのサーバでポリシーの削減ができていることが確認できる．特に Apache では削減率が高い．これは，Apache ではサーバの初期化時にモジュールの組み込みや，ライブラリのリンク，設定ファイルを読み込み等を行い，HTTP 処理フェーズではサーバ本来の処理のみを行うからである．それに対して sendmail, dovecot では削減率がやや低い．

表 1: ポリシーのルール数の比較

	Apache	sendmail	dovecot
strict ポリシー	150	164	183
提案手法	61	140	143
削減数	89	24	40
削減率	59.3%	14.6%	23.5%

表 2: ポリシーのタイプ数の比較

	Apache	sendmail	dovecot
strict ポリシー	39	34	37
提案手法	16	34	30
削減数	23	0	7

その原因はプロトコル処理フェーズでの挙動にある．sendmail では，メールキューへのアクセスや，メールの転送，負荷に応じた動作をするための/proc へのアクセス等を SMTP 処理フェーズで行う．また，dovecot では，ユーザ認証を行う際に動的にライブラリを読み込む構造になっているため，POP 処理フェーズでも頻繁にライブラリにアクセスする．以上の理由により，Apache に比べて削減率が低くなっている．

表 2 より，Apache では 23 個のタイプが削減できていることが確認できる．削減できたタイプは，httpd_module_t, sbin_t, device_t, net_conf_t, bin_t 等のサーバの初期化のみで必要とするタイプである．sendmail ではタイプを削減することができなかった．dovecot では，初期化のみで必要とする devpts_t, dovecot_etc_t 等が削減されている．

表 3 は削減されたアクセスベクタのうち，多く削減されたもの，三種類のサーバに共通して削減されたものを表している．この表より，Apache では多くの read が削減できていることがわかる．read の対象となるタイプは，httpd_modules_t, lib_t, httpd_conf_t 等であった．さらに，Apache, sendmail, dovecot に共通して，bind や listen 等のネットワークの確立の前処理に必要なアクセスベクタが削減されていることが確認できる．

5.2 フェーズ遷移のオーバーヘッド

実際に記述した HTTP，SMTP，POP 処理フェーズのポリシーを用いて，フェーズ遷移処理に要するオーバーヘッドを測定した．結果，オーバヘッドは Apache では 846 ミリ秒，sendmail では 842 ミリ秒，dovecot では 898 ミリ秒であった．つまり，初めて外

表 3: 削減されたアクセスベクタの例

	Apache	sendmail	dovecot
read	23	2	9
getattr	18	7	2
search	12	0	2
create	5	1	1
ioctl	5	1	0
write	4	3	3
connect	2	0	1
listen	1	1	2
bind	1	1	2
name_bind	1	1	1
node_bind	1	1	1
net_bind_service	1	1	1

部との接続が確立したときに 900 ミリ秒程度の遅延が生じる. フェーズの遷移は基本的に一回しか行われないので, このオーバーヘッドは許容できるものと言える.

6 関連研究

品川ら [12] は, サンドボックスにおける実行時のフェーズを考慮したポリシー記述の簡略化を提案している. 本論文はこの研究を基礎とし, セキュア OS に応用したものである.

Zanin ら [11] は SELinux のポリシー設定を分析するための SELAC と呼ばれるフレームワークを提唱している. SELinux のポリシーの Type Enforcement やドメイン遷移等のルールを記号化し, Jaeger ら [6][7] のサブジェクトに対して与えられたパーミッションの状態を表すアクセス制御空間の概念を利用してどのサブジェクトがどのオブジェクトへのアクセスを許可されているかを確認するためのアルゴリズムを開発している. この研究ではポリシーが正しく設定されているかを分析することが目的であり, ポリシーの記述量は変わらない.

setools[10] は, Tresys Technology 社が開発した SELinux 用設定支援ツールである. GUI を利用して, ポリシーの分析, 設定を行うことができ, 広く使われている. しかし, ポリシーの記述量は変わらない.

ポリシーの動的切り替えを一般化した概念として, オートマトンを用いた状態遷移に基づくセキュリティモデル [3][8] がある. これらの研究は情報の流れに着目しているという点では本研究と似ているが, その目的は主に情報漏洩の防止であり, ポリシーの簡略化を提供するものではない.

7 まとめ

外部からの攻撃を対象としたインターネットサーバにおけるセキュア OS のポリシー記述の簡略化の手段として, 実行時のフェーズを考慮したポリシー記述の簡略化を提案した. 提案手法では, サーバが外部と接続を確立するまではリモート攻撃は起こりえない点に着目し, システムが起動してからサーバが外部と接続を確立するまでの初期化フェーズと, 接続を確立した後のプロトコル処理フェーズとを考え, 初期化フェーズではアクセス制御を行わず, プロトコル処理フェーズでのみアクセス制御を行う. これにより, サーバがサービスを開始するまでの前処理に必要なポリシー記述を簡略化できる. 実際に HTTP, SMTP, POP の三種類のサーバに対して提案手法を適用し, ポリシー記述の簡略化を評価したところ, 各サーバに対して記述した strict ポリシーに比べて, それぞれ 59.3%, 14.6%, 23.5%のルール数削減につながった.

参考文献

[1] National Security Agency. Security-enhanced linux. `http://www.nsa.gov/selinux/`.

[2] W.E. Boebert and R.Y. Kain. A practical alternative to hierarchical integrity policies. In *Proceedings of the 8th National Computer Security Conference*, 1985.

[3] U. Erlingsson and F.B. Schneider. Sasi enforcement of security policies. In *Proceedings of the 1999 Workshp on New Security Paradigms(NSPW '99)*, pp. 87–95, 1999.

[4] Red Hat. Strict policy. `http://www.redhat.com/`.

[5] Red Hat. Targeted policy. `http://www.redhat.com/`.

[6] Trent Jaeger, Antony Edwards, and Xiaolan Zhang. Managing access control policies using access control spaces. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pp. 3–12, New York, NY, USA, 2002. ACM.

[7] Trent Jaeger, Xiaolan Zhang, and Antony Edwards. Policy management using access control spaces. *ACM Trans. Inf. Syst. Secur.*, Vol. 6, No. 3, pp. 327–364, 2003.

[8] N.V. Mehta and K.R. Sollins. Expanding and extending the security features of java. In *Proceedings of the 7th USENIX Security Symposium*, pp. 159–172, 1998.

[9] J.H. Saltzer and M.D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, Vol. 63, No. 19, pp. 1278–1308, 1975.

[10] Tresys Technology. Setools. `http://oss.tresys.com/projects/setools`.

[11] Giorgio Zanin and Luigi Vincenzo Mancini. Towards a formal model for security policies specification and validation in the selinux system. In *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, pp. 136–145, New York, NY, USA, 2004. ACM.

[12] 品川高廣, 河野健二. 実行時のフェーズを考慮したセキュリティポリシー記述の簡略化. 先進的計算基盤シンポジウム SACSIS2006, pp. 495–503, 2006.