情報処理学会論文誌 Vol.53 No.9 1–18 (Sep. 2012)

アプリケーションの実行状況に基づく強制アクセス制御方式

原田 季栄[1,2,a)] 半田 哲夫[3,b)] 橋本 正樹[1,c)] 田中 英彦[1,d)]

受付日 2011年12月2日, 採録日 2012年6月4日

概要：従来のアクセス制御は，主体であるアプリケーションとそれがアクセスしようとするファイルなどの客体の組み合わせによってアクセス可否を判断していた．そのためアプリケーションの処理の内容およびアクセスを認めることにより情報システムに与える影響を考慮することができなかった．本稿では，アプリケーションが実行される状況に基づき，各アプリケーションが行おうとしている処理の内容を考慮することができるアクセス制御方式について提案する．提案方式を用いることにより，不正アクセスや誤操作などによるリスクを軽減することが可能となる．本稿では，提案システムの概念と実現方法について紹介し，その Linux 上の実装である TOMOYO Linux[1][2] における評価結果を報告する．

Mandatory Access Control Method Based on Application Execution State

Toshiharu Harada[1,2,a)] Tetsuo Handa[3,b)] Masaki Hashimoto[1,c)] Hidehiko Tanaka[1,d)]

Received: December 2, 2011, Accepted: June 4, 2012

Abstract: Existing access control methods grant access requests based on the combinations of applications as subject and files as objects. Therefore intents of applications and the possible effects caused by granting the access requests have not been taken into consideration. In this paper, we propose a new access control method based on application history and intents. With our access control method, system administrators can reduce the risks caused by malicious access attempts and wrong operations. In this paper, the concept and implementation design will be explained as well as the brief evaluation report of TOMOYO Linux, our implementation of the new access control method to Linux.

1. はじめに

近年，情報システムの適用領域が拡大し，その処理内容も高度化しているため，それを実現するソフトウェアは大規模・複雑化している．そのため，情報セキュリティの担保が課題となっており，例えば，ファイルやディレクトリのアクセス権の設定が適切でなければ，情報の漏えいやシステムの不具合が容易に生じ得るし，それらが適切であったとしても，不正アクセス，ワームなどにより情報システムが乗っ取られてしまうリスクが潜在的に存在する．これらのリスクは，完全に解消させることは不可能であり，ソフトウェアの規模が大きくなり，アプリケーションの処理内容が複雑になることにより，今後確実に増大する．

情報セキュリティを担保する手法として，サンドボックス [3][4] とアクセス制御がある．サンドボックスは，アプリケーションを限定された環境で実行するもので，ネットワークからダウンロードしたファイルなど信頼できない

[1] 情報セキュリティ大学院大学
INSTITUTE of INFORMATION SECURITY
[2] 株式会社 NTT データ
NTT DATA CORPORATION
[3] NTT データ先端技術株式会社
NTT DATA INTELLILINK CORPORATION
[a)] dgs085101@iisec.ac.jp
[b)] penguin-kernel@i-love.sakura.ne.jp
[c)] hashimoto@iisec.ac.jp
[d)] tanaka@iisec.ac.jp

© 2012 Information Processing Society of Japan

アプリケーションについて，他と隔離された環境（サンドボックス）の中で実行させることにより，アプリケーションを閉じ込め [5]，被害がおよぶ範囲を限定することができる．また，アクセス制御は，サンドボックスを実現するための要素技術で，アプリケーションの実行に伴い発生するアクセス要求について，その実行可否を判定するものである．情報システムは，アプリケーションがファイルの読み書き，他のアプリケーションの実行，ネットワーク通信などを要求し，カーネルがそれを処理することにより機能する．したがって，個々のアプリケーションのアクセス要求を適切に制御することにより，システム管理者が望まない情報システムの利用を防ぐことができる．

従来のアクセス制御方式では，アプリケーションとそれがアクセスしようとする対象ファイルの組み合わせに基づいてアクセス可否を決定していた．アクセスを要求しているアプリケーションの実行状況については考慮されておらず，それをアクセス制御の条件（判断基準）とできないため，情報セキュリティを損なう事態が生じていた．例えば，Web ブラウザ（HTTP クライアント）などのように多種多様な処理を行うアプリケーションは，本来はその実行状況により認めるアクセスを変えるべき [6] であるが，従来のアクセス制御方式では，それを考慮した実行可否を定義することができない．結果として，従来のアクセス制御方式に基づくサンドボックスは粒度が粗く，被害範囲の局所化や封じ込めが不十分であった．

そこで本稿では，情報セキュリティを担保するための基礎技術であるアクセス制御について，アプリケーションの実行状況を考慮可能な新しいアクセス制御方式について提案する．提案方式では，アプリケーションの実行状況を，システム起動時以降から該当アプリケーションの実行に至るまでの履歴と，アプリケーションのコマンドライン引数や要求発生時の環境変数等の情報から解釈し，これらの情報を条件に用いることによりアクセス可否を判定する．また，提案方式では，情報の取得／判定／強制をカーネルから行うので [7]，安全に漏れなくアクセス制御の強制を行うことができる．

本稿の構成について説明する．2 章では，従来のアクセス制御方式について振り返り，その課題について述べる．次に，3 章では，提案方式を実現するための基本的なアイデアについて述べる．また，4 章では，提案方式の Linux オペレーティングシステム（以下 OS）に対する実装として，筆者らの開発している TOMOYO Linux を取り上げて説明し，ポリシー記述例により提案方式が可能とするアクセス制御の例を示す．5 章では，本方式の評価として，本方式の特長と使いやすさに関する初期検証の結果，性能への影響について述べる．さらに，6 章では，提案方式と他の方式との比較，不正アクセスに対する耐性と検討課題について考察を行う．最後に，7 章では，本稿の提案内容をまとめ，結論を述べる．

2. 従来のアクセス制御方式の限界と課題

本章では，はじめに，今日多くの OS で用いられている二つのアクセス制御方式を中心に，既存研究について振り返り，本稿で用いるアクセス制御に関係する基本的な用語について整理する．その後，従来方式の限界について述べ，解決すべき課題を整理する．

2.1 従来のアクセス制御方式について

今日，多くの情報システムで用いられるアクセス制御方式は，自由裁量のアクセス制御 (DAC: Discretionary Access Control) と強制アクセス制御 (MAC: Mandatory Access Control) の 2 つに大別される [8]．DAC は，ファイルなどの所有者がそれに対してアクセスできるユーザを制御するもので，アイデンティティ情報を用いてアクセス可否を判定することから，アイデンティティに基づくアクセス制御 (identity-based access control) とも呼ばれる．一方で，MAC は，所有者であってもアクセスできるユーザを制御できないもので，システム管理者が定めたルールに基づき個々のアクセス可否を判定する．MAC はアクセス可否のルールを必要とすることから，ルールに基づくアクセス制御 (rule-based access control) とも呼ばれる．

MAC には，アクセス可否を判断する際の基準となるルールが必要であり，それは一般にポリシーと呼ばれる．ポリシーでは，アクセス制御の対象となる主体および客体に何らかの識別子を必要とする．1983 年にアメリカ国防総省が発表した TCSEC (Trusted Computing Systems Evaluation Criteria)[9] で定義されたラベルに基づく MAC(Labeled Security) では，この識別子を「ラベル」と呼んでいる．TCSEC 以降，MAC の実装はラベルに基づく MAC しかなかったが，2006 年よりパス名を識別子として用いる強制的なアクセス制御が提案され，それらはパス名に基づく MAC(pathname-based MAC)[*1] と呼ばれている．例えば，Linux カーネルの標準ソースコードには，SELinux[10][11][12]，SMACK[13], TOMOYO Linux, AppArmor[14] の 4 つの MAC の実装が現在含まれており，このうち，SELinux と SMACK はラベルに基づく MAC に該当し，TOMOYO Linux と AppArmor はパス名に基づく MAC となる．

ポリシーは，許可されるものと許可されないものを記述した条件の集合体と定義される．ポリシーを解釈し，それを実現するものは，ポリシーメカニズムと呼ばれる．アクセス制御方式の議論では，一般に主体（Subject）と客体（Object）という言葉が用いられる．本稿もそれにしたがう．ここまでの説明では，「アプリケーション」について，

[*1] http://lwn.net/Articles/277833/

© 2012 Information Processing Society of Japan

広くコンピュータプログラムを指して用いていたが，本論に入るにあたり言葉の定義を明確にしておく．これ以降，本論文における「アプリケーション」は，OS のプロセス実行命令（Linux では execve システムコール）が認識する実行形式ファイルを指して用いる．アプリケーションの実行単位，あるいはインスタンスについては，プロセス（OS プロセス）と呼ぶ．

上記の各種 MAC が，カーネル内で自動的に（受動的に）適用されるのに対して，アプリケーション自身の申告に基づいたアクセス制御を行う方式として，Linux の seccomp[15], FreeBSD の Capsicum[16] がある．seccomp では，プロセスが自ら prctl(PR_SET_SECCOMP, 1); というシステムコールを発行すると，それ以降, read(), write(), exit(), sigreturn() の 4 つのシステムコール以外のシステムコールを実行できなくなる．現状の seccomp はグリッドなど特殊な用途にしか適用できないが，利用できるシステムコールを汎用化する提案がなされている．Capsicum ではアプリケーションごとにプロセスのケーパビリティを制限することが可能であり，seccomp と同様にプロセスの申告に基づく．Capsicum では，アプリケーションごとにオープンできるファイルを制御するといった用途には適用できないので，直接本研究の課題を単独で解決することはできない．

2.2 従来のアクセス制御方式の限界

DAC は，もっとも基本的なアクセス制御方式として広く使われている．DAC では，ユーザ情報に基づいてアクセス可否を判断するため，定義できるアクセス許可の内容は読み書き実行と粗い上に，管理者権限を持つユーザは影響を受けないという限界がある．このため，バッファオーバーフロー攻撃 [17] などにより制御を奪われてしまった場合，被害を限定する効果が弱く，特に管理者権限を持って実行されているプロセスの制御を奪われた場合には，深刻な被害を避けられない．

ラベルに基づく MAC は，DAC における限界を補うことができる．ラベルに基づく MAC では，主体と客体に割り当てられたラベルに基づきアクセス可否を判断する．必要に応じて，主体および客体のラベルを定義すれば，任意の粒度でのアクセス制御を行うことが可能であり，管理者権限で実行されているプロセスも例外扱いされず制御の対象となる．しかしながら，ラベルに基づく MAC を用いることにより，主体から客体へのアクセス可否の判断を適切に制御できたとしても，それだけでは情報セキュリティを保つことができるとは言えない．それについて二つの例を示す．

一つ目の例として，Web サーバの Apache は，コンテンツが保存されているディレクトリに.htaccess という名前を持つファイルが存在する場合，そのファイルの内容を設定として解釈し，Web コンテンツとしては扱わない（クライアントに送らない）．しかし，そのファイルが index.txt という名前にリネームされるとその内容はコンテンツとしてクライアントに配信されてしまうことになる．ラベルに基づく MAC を用いて，これを防ぐためには，設定ファイルである.htaccess と，index.txt などコンテンツのファイルとで異なるラベルが割り当てられており，かつ Web サーバの Apache のプロセスがファイルをオープンした後にオープンされたファイルのラベルを取得し，そのラベルの内容に応じて，それをどのように扱うかを決めるように実装されていなければならない．しかし，現状ではどのファイルにどのラベルを割り当てるかについては，共通の規約や同意は存在しておらず[*2]，ほとんどのアプリケーションはファイル名に基づいて動作や扱い方を決めるように実装されている．

二つ目の例として，SSH（セキュアシェル）サービスのサーバは，クライアントからの要求受付時に，指定されたファイルの内容をバナーとして表示するオプションを持つ．例えば，/usr/sbin/sshd -o 'Banner /etc/shadow' のように実行すると，認証を行う前のクライアントに対して /etc/shadow に保存されているパスワードを開示してしまうことになる．一つ目の例は，ファイル名がアプリケーションの処理内容に影響を与える例で，二つ目の例は，アプリケーションに与えられたコマンドライン引数がアプリケーションの処理内容を決定する例で，それらは従来のアクセス制御では対処することができない．

ラベルに基づく MAC は識別子として名前（ファイル名）を用いない．それは，名前が主体および客体の属性であり変わり得るものであることが最大の理由である．実体にラベルを割り当てることによって，リネームなど名前に関する操作に影響されないアクセス制御が可能となり，客体の情報の漏えいを防ぐことができるが，ラベルが維持され，客体へのアクセス条件が保たれたとしても，それだけでは十分ではない．

先に見た二つの例は，それぞれ Apache と sshd というアプリケーションの処理内容に関するものであり，かつファイルのリネームおよびアプリケーションの実行というアクセスを許可したことによる生じる影響である．従来のアクセス制御では，それらを考慮していないので，制限することができない．

2.3 解決すべき課題

従来のアクセス制御方式は，主体と客体の組み合わせに基づいてアクセス可否の判断を行っており，主体であるアプリケーションについてはそれがどのような処理を行おうとしているかは考慮されていなかった．2.2 節で示した一

[*2] Fedora15 においては，/var/www/html の配下に.htaccess を作成した場合，コンテンツと同じラベルが割り当てられる

© 2012 Information Processing Society of Japan

つ目の例では，客体に保存されていた情報がどのように使われるようになるかが考慮されないまま，リネームの要求を認めていることが問題であり，二つ目の例では，認証前のクライアントにパスワードファイルの内容を表示してしまうことになるということが考慮されないまま，SSH サーバの実行を認めていることが問題である．すなわち，従来のアクセス制御方式には以下の課題が存在する．

- アクセス要求の可否判断において，客体に保存されていた情報がどう使われるのかが考慮されていない
- アクセス要求の可否判断において，情報システムやアプリケーションの動作にどのような影響が生じるかが考慮されていない

アクセス制御機構において，以下の仕組みを取り込むことで上記の課題を解決することができる．

(i) アプリケーションごとに，その実行されている状況を取得（分類）する仕組み

同じアプリケーションであっても，それが実行されている状況によって，必要となるアクセス許可定義は異なる．そのため，何らかの基準を用いてアプリケーションの実行されている状況を分類し，その分類ごとにアクセス制御を行うことが必要となる．

(ii) アプリケーションについて，それが行おうとしている内容を判断する仕組み

それぞれの状況ごとに分類ができたとして，それぞれのアプリケーションから発せられるアクセス要求について，その意図を推測し，アクセスを許可した場合の影響を判断するための基準が必要である．

(iii) すべてのアプリケーションについて，その状態と行おうとしている内容を考慮したアクセス可否判断を行う仕組み

上記 (i)(ii) に基づいたアクセス制御機構は，それが漏れなく，回避できないように実装される必要がある．

3. 実行状況に基づくアクセス制御方式の提案

本章では，第 2 章で述べた従来のアクセス制御の限界を補う，新しいアクセス制御を実現するための基本的なアイデアとして，実行履歴を用いたアプリケーションの分類と，その処理内容によるアクセス可否の判定について説明する．

3.1 実行履歴を用いたアプリケーションの分類

アプリケーションごとにその処理内容を考慮したアクセス制御を行うためには，アクセス制御機構において，アプリケーションを意識しなければならない．同じアプリケーションであっても，それが実行される状況によって扱いを変える必要がある．たとえば，サーバのコンソールからログインして得られたシェル（Linux におけるコマンドインタプリタ）と，SSH クライアントからログインして得られたシェル，あるいは Web サーバの Apache が CGI を実行するために得たシェルは，プログラムファイルとしては同一であっても与えるべきアクセス許可は区別して扱うべきである．また，本稿の対象とするアクセス制御は，情報セキュリティを保つものであり，すべてのアプリケーションを対象とすることが望ましい．すなわち，すべてのアプリケーションについて，その実行状況により場合を分離できるようにする必要がある．本稿では，アプリケーションごとにその実行履歴を管理することにより，アプリケーションの実行状況を分類する方法を提案する．

ここで実行履歴とは，アプリケーションのプロセスごとに，それを実行するに至ったアプリケーションのプログラム名をリスト化したものである．**表 1** に，Linux ディストリビューションのひとつである Fedora 15 における例[*3]を示す．3 つの例は，いずれもシェルである/bin/bash のプロセスに関するものである．項番 1 から 3 までの先頭にある/sbin/init は，Linux 起動時に作成されるプロセスであるが，そこから/bin/bash が実行されるにいたった履歴は 3 つとも異なっている．項番 1 では，サービスを起動する起動用スクリプトである/etc/rc.d/init.d/sshd から実行されたデーモンである sshd が，クライアントからのアクセス要求により新たなプロセスを生成し，そこから実行された/bin/bash であることを示している．項番 2 では，ログインを受け付けるアプリケーションである/sbin/agetty が，ログイン処理を行うためのアプリケーションである/bin/login を実行し，そこから実行された/bin/bash であることを示している．項番 3 では，項番 2 として実行された/bin/bash が受け付けた「他のユーザとしてコマンドを実行するためのアプリケーション」である su(switch user) の実行結果として，実行された/bin/bash であることを示している．

Linux カーネルは標準の状態では，3 つの例/bin/bash のプロセスについて，それらがどのような状況で実行されたものであるかを判定することができない．したがってそれらの置かれた状況により区別することができず，それぞれのプロセスについて状況に応じたアクセス許可を定義することもできない．しかし，各プロセスについて，それぞれの実行履歴の内容を参照することができれば，コンソールからログインした場合には制限なくすべてのアプリケーションを実行できるが，SSH クライアントからログインした場合には特定のアプリケーションしか実行できないように制限するなど，それぞれに認めるアクセス許可を区別することができる．

3.2 アプリケーションの処理内容によるアクセス可否判定

アプリケーションの処理内容は，アプリケーションの作者が定義し，アプリケーションのファイルの中に記録され

[*3] これらの結果は，ディストリビューションや実行環境により異なる．

© 2012 Information Processing Society of Japan

表 1 実行履歴の例 (Linux)
Table 1 Examples of Program Execution History (Linux).

1	SSH サービスからログインして実行されている bash プロセス /sbin/init /etc/rc.d/init.d/sshd /usr/sbin/sshd /usr/sbin/sshd /bin/bash
2	コンソールからログインし得られた bash プロセス /sbin/init /sbin/agetty /bin/login /bin/bash
3	コンソールからログインし，管理者として作業するため su コマンドを実行して得られた bash プロセス /sbin/init /sbin/agetty /bin/login /bin/bash /bin/su /bin/bash

ている．したがって，アプリケーションの処理内容を完全に把握するためには，アプリケーション自身にそれを申告させるのが最上であるが，すべてのアプリケーションに適用できる基準や記述方法は現状存在していない．また，アプリケーションが乗っ取られる可能性を考えると，権限を減らすことには問題ないが，権限を増やす申告を認めることは新たなリスクとなる．仮にそれらの問題が解決できた場合にも，既存のアプリケーションの修正が必要となる．このような背景により，直接的な手法によって，アプリケーションの処理内容を判定することは困難であるが，間接的手法を用いることにより，一定の範囲でアプリケーションの修正を必要とすることなく，その処理内容を判定することが可能である．

アプリケーションには，処理内容に影響を与える要因が存在する．それらをアクセス許可を与える際の条件（パラメータ）として用いることができれば，不適切な要求を識別，拒否することができる．アプリケーションの処理内容に影響を与える要因について，いくつかの具体例を示す．

- コマンドライン引数（多くのアプリケーションはオプション指定を受け付ける）
- 環境変数（実行時に参照するコマンドサーチパス，プロキシーサーバなど）
- ユーザ入力（ユーザの指示により行う処理を分岐する）
- 入力データ（データの内容を解析して結果を表示する）
- 設定ファイル（多くのアプリケーションは独自の設定ファイルを持ち，起動時に参照する）
- ライブラリ（故意に違うライブラリを参照させる攻撃が存在する）
- 時間
- 実行しているユーザ（管理者の場合処理を行う等）
- 特定の名前を持つファイルの有無や内容（/etc/nologin, .htaccess 等）

注目すべきことは，上記を含む情報はすでにカーネル内で管理されているので，MAC の処理の中でそれをパラメータとして参照することが可能である．本稿では，MAC の中でアプリケーションの処理内容に影響を与える情報をポリシーの中でアクセス可否の条件基準パラメータとして用いることを提案する．

4. Linux に対する提案方式の実装

本章では，アプリケーションの処理内容を考慮したアクセス制御方式の実装について，TOMOYO Linux を例に説明する．TOMOYO Linux は Linux 上の MAC の実装例，およびそれを推進するプロジェクトの名称である．TOMOYO Linux の方式および機能の検討については，著者のうち原田と半田が行い，提案方式の Linux カーネルへの実装とポリシーエディタなど標準ツールのコーディングについては，半田が単独で行っている．TOMOYO Linux は，提案方式の検討を受けて随時拡張が行われており，本論文執筆時点における最新版であるバージョン 1.8.3 は本論文の内容に対応している．TOMOYO Linux のソースコードについて，SourceForge.jp 上[*4]で公開している．Linux カーネルのバージョン 2.6.30 以降には，メインライン向けに作成された機能限定版の TOMOYO Linux が標準機能として含まれている．

4.1 アプリケーションの実行状況の取得

Linux カーネルは標準の状態では，実行するアプリケーションがどのような履歴を持つかを管理していない．各アプリケーションのプロセスは，自分をつくりだした親のプロセスの id を覚えているが，終了したプロセスの情報はカーネルから削除されるため，限られた範囲でしか親プロセスをたどることができない．

もし，すべてのアプリケーションのプロセスについて，共通の基点から自身のプロセスが生成されるにいたった履歴を持たせ，それらをプロセスの終了後も保持することができれば，各アプリケーションの実行状況を一意に定めることが可能となる．

4.1.1 「プログラム実行履歴」概念の定義と実現方式

TOMOYO Linux では，アプリケーションの実行状況の取得を可能とするために，プログラム実行履歴の概念を実現した．プログラム実行履歴とは，各プロセスが作られるまでに実行されたアプリケーションのパス名と実行しているアプリケーションのプログラム名（パス名）を半角スペース文字をセパレータとして並べたものである．先に見た表 1 の内容は，そのまま/bin/bash に関するプログラム

[*4] http://tomoyo.sourceforge.jp/

© 2012 Information Processing Society of Japan

実行履歴の例となっている．

Linux におけるアプリケーション実行の仕組みを利用することにより，すべてのプロセスについてそのプログラム実行履歴を持たせることができる．その方法について図を用いて説明する．**図 1** は，Linux でログインして得られた /bin/bash から，/bin/date（日付の表示と設定を行うコマンド）を実行して，その結果を待ちあわせる際に発行されるシステムコール（アプリケーションがカーネル機能を呼び出すためのインタフェース）の内容を示している．Linux など UNIX の流れを汲む OS では，アプリケーションの実行について特徴的な手順により行う．まず，アプリケーションを実行しようとする（親となる）プロセスが fork() システムコールにより自身の複製となるプロセスを生成し，複製されたプロセスは execve() システムコールを呼び出すことによってその内容（プロセスイメージ）がアプリケーションのものに置き換わる．図 1 で，/bin/bash のプロセスは fork() システムコールを発行し，その複製されたプロセスが execve() システムコールを発行することにより/bin/date が実行されている（/bin/date のプロセスが終了すると/bin/bash のプロセスに処理が戻る）．各プロセスの右側に表示されているのは，それぞれのプロセスのプログラム実行履歴である．その内容はプロセスの生成時に親のプロセスから引き継がれ（複写され），execve() を行う際に実行しようとするアプリケーションのプログラム名を追加することにより更新できることがわかる．

4.1.2 ドメイン

TOMOYO Linux では，ドメインという単位でアクセス制御を行う．TOMOYO Linux におけるドメインについて述べる．ドメインは，先に述べたプログラム実行履歴について，その基点を追加したものである．アプリケーションが何も実行されていない状態である「カーネル自身（カーネルスレッド）」について，そのプログラム名を仮想的に<kernel>と表記することにする．定義によりすべてのドメインは必ず<kernel>で始まるので，<kernel>は共通の基点となる．基点を導入することにより，プログラム実行履歴の一部と全体の区別が可能となる．

ドメインを指定することによりプログラム実行履歴の内容が一意に定まる．あるプロセスについて，そのプログラム実行履歴がドメインと一致するとき，プロセスはそのドメインに属していると呼ぶ．Linux で最初に作られるプロセスである/sbin/init は，<kernel> /sbin/init のドメインに属する．コンソールからログインして得られた /bin/bash の所属するドメインは，<kernel> /sbin/init /sbin/agetty /bin/login /bin/bash となる．

ドメインは以下の性質を持つ．

- すべてのドメインは親となるドメインをただ一つ持つ
- すべてのプロセスはひとつのドメインに所属する
- 実行されているアプリケーションが同一であっても同じドメインに所属するとは限らない

TOMOYO Linux では，すべてのプロセスについて，プログラム実行履歴を記憶しておき，その属するドメインごとにアクセス許可を定義し，それに基づきアクセス制御を行う．**図 2** は，Fedora 15 における TOMOYO Linux のポリシーエディタの画面のコピーである．各行がそれぞれ異なるドメインに対応しており，現在までに実行されたアプリケーションのプログラム名が表示されている．行間のインデントはドメインの親子関係を示しており，同じインデントを持つドメインは共通のドメインのプロセスから生成されたものであることを示す．

4.2 アプリケーションの処理内容の判定

漏れなく強制的なアクセス制御はカーネルで行う必要がある．カーネルはアプリケーションからの断片的な要求について，それらを処理するという分担であるため，アプリケーションからの要求について，その意図や複数の要求の関連性を判定することはできない．アプリケーションの意図や要求の関連性を知るのは，アプリケーションの作者であり，アプリケーションのプログラム自身である．仮にアプリケーションからカーネルに対してその意図を伝えるようにしたとしても，アプリケーションが乗っ取られた場合を考慮するとその申告を信用することはできない．また，それを行うためには既存のプログラムの修正と再コンパイルが必要となるデメリットもある．しかし，すでにカーネル内に存在する情報（カーネルが管理している情報）であれば，アプリケーションに申告させる必要がないため，既存のプログラムの修正を必要とせず，またアプリケーションの乗っ取りによる影響を受けず信用することができる．

Linux カーネル内には，プロセスに関する情報，i ノード（ファイルの実体）に関する情報，アプリケーション実行要求に関する情報，ファイルに関する情報などが存在しており，ポリシー言語の仕様で，それらの情報に対応する変数を定義し，ポリシー言語の構文で条件式として，カーネル内のパラメータをアクセス制御の条件として記述できるようにした．

4.3 MAC の実装

漏れなく，回避できないアクセス制御を行うためには，リファレンスモニタ [18][19] を実装すれば良い．具体的には，システムコールの呼び出しを捕捉（フック）し，その要求可否の判断を行う．許可する場合には本来の処理を実行するが，そうでない場合は要求を拒否し，呼出しもとにエラーを返す．リファレンスモニタの実装はフックする関数の書き換えにより行うが，バージョン 2.6 以降の Linux は，カーネルにおけるアクセス制御の処理内容を拡張するためのフレームワークである Linux Security Modules[20]（以下，LSM と略す）を備えている．LSM を用いると，直

© 2012 Information Processing Society of Japan

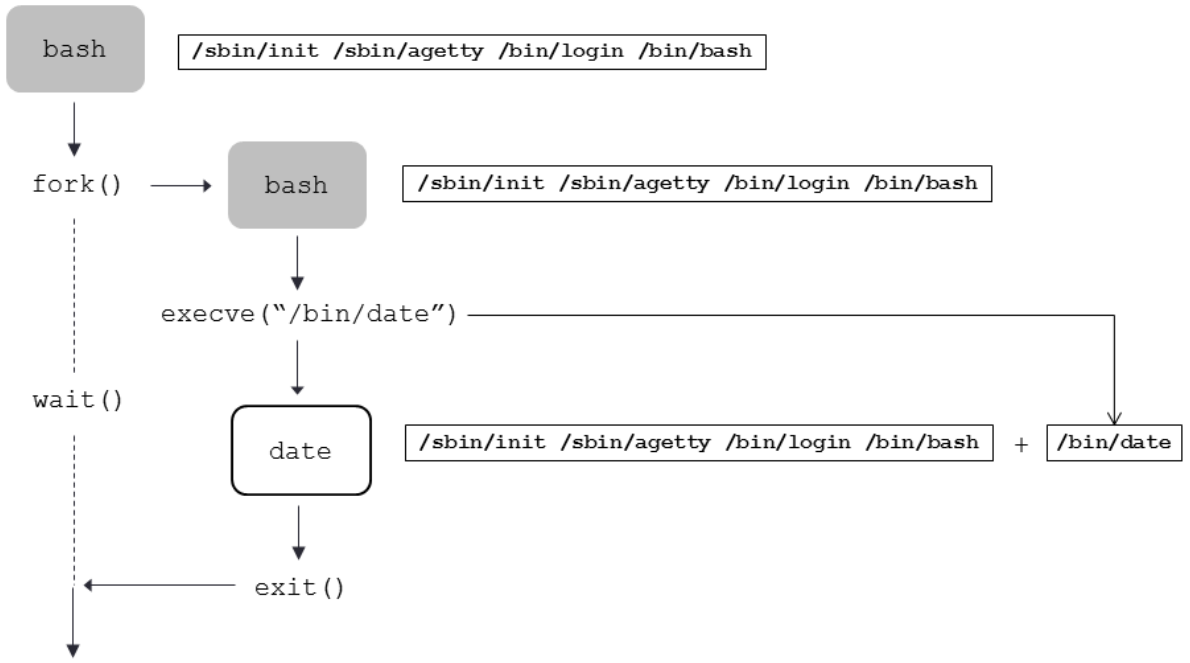


図 1 プログラム実行履歴の定義
Fig. 1 Defining Program Execution History.

```
<<< Domain Transition Editor >>>      383 domains    '?' for help

<kernel> /sbin/init /bin/dbus-daemon
    0:  1     <kernel>
                  => <kernel> /sbin/modprobe ( -> 381 )
    1:  1         /lib/systemd/systemd-cgroups-agent
    2:  1         /sbin/init
                      => <kernel> /sbin/modprobe ( -> 381 )
    3:  1             /bin/dbus-daemon
    4:  1                 /lib/dbus-1/dbus-daemon-launch-helper
    5:  1                 /usr/libexec/colord
    6:  1                 /usr/libexec/fprintd
    7:  1                 /usr/libexec/nm-dispatcher.action
    8:  1                     /etc/NetworkManager/dispatcher.d/00-netreport
    9:  1                         /bin/mountpoint
   10:  1                         /sbin/consoletype
   11:  1                     /etc/NetworkManager/dispatcher.d/04-iscsi
   12:  1                         /bin/grep
   13:  1                         /sbin/chkconfig
   14:  1                             /bin/sh
   15:  1                                 /sbin/runlevel
   16:  1                         /sbin/ip
```

図 2 ドメイン遷移の例 (Fedora 15)
Fig. 2 Domain Transition Example (Fedora 15).

© 2012 Information Processing Society of Japan

接システムコールの処理内容を書き換えることなく，それをフックし，定義した処理（ここではアクセス制御）を追加することができる．LSM ではシステムコールフックの内容について，「モジュール（セキュリティサーバとも呼ぶ）」と呼ぶ単位で管理しているが，LSM の現在の仕様は複数のモジュールの併用を認めていない．TOMOYO Linux は，他のモジュールとの併用を可能とするため，LSM を使用したものと，独自にシステムコールをフックするバージョンを提供している．

4.4 ポリシー仕様

本節では，TOMOYO Linux を用いることにより，従来のアクセス制御方式では，記述することができなかった多様な条件を考慮したアクセス制御が行えることを具体的なポリシー例により示す．TOMOYO Linux は，ファイルの他にネットワークやシグナル送信などのアクセス制御に対応しているが，紙面の都合によりファイルに関するアクセス許可についてのみ説明する．

アクセス許可は，「カテゴリ　操作　必須条件　任意条件」という構成を持つ．「カテゴリ」はアクセス制御の分類であり，ファイルに関するアクセス許可のカテゴリは `file` である．「カテゴリ　操作　必須条件」は必須で，「操作」は客体に対して行おうとしている操作で，ファイルについては `rename`, `execute` などがある．「必須条件」は操作の対象となるファイルのパス名などである．パス名については，`/tmp` 配下にプロセス ID を含む名前で作られる一時ファイルなど，名前が事前に決まらないものがある．また，Web コンテンツなど，特定のディレクトリ配下のすべてのファイルを列挙せずまとめて許可したい場合がある．それらの場合について，TOMOYO Linux ではワイルドカードを用いることで対応している．**表 2** に，TOMOYO Linux で利用できるワイルドカードの一覧を示す．

「条件」については，必須条件と任意条件の 2 種が存在する．必須条件は，「操作」の内容に対応して必ず指定しなければならないもので，必要となる条件の内容は「操作」ごとに異なる．たとえば `file rename` を許可するためには，必ず「リネーム前の名前」と「リネーム後の名前」を指定しなければならず，その 2 つは必須条件である．

任意条件は，必要に応じてアクセスを許可する条件をより厳しく指定するために用いるもので，すべてのアクセス許可に対して，任意の数の任意条件を指定することができる．

以上 TOMOYO Linux ポリシーの概要について説明した．ポリシー仕様の詳細については，プロジェクトの Web ページ[*5]で公開している．

[*5] `http://tomoyo.sourceforge.jp/1.8/policy-specification/index.html`

表 2 TOMOYO Linux で利用できるワイルドカード
Table 2 TOMOYO Linux wild card patterns.

パターン	意味
`\*`	/ 以外の 0 回以上の繰り返し
`\@`	/ と . 以外の 0 回以上の繰り返し
`\?`	/ 以外の 1 文字
`\$`	1 桁以上の 10 進数
`\+`	10 進数 1 桁
`\X`	1 桁以上の 16 進数
`\x`	16 進数 1 桁
`\A`	1 文字以上のアルファベット
`\a`	アルファベット 1 文字
`\-`	パス名を除外する演算子
`/\{dir\}/`	1 回以上の dir/の繰り返し

4.4.1 ドメインポリシーの例

TOMOYO Linux で，コンソールからログインしたユーザが自分のパスワードの変更を行うために必要なポリシーについて例を示す．この例では，まずコンソールからログインして実行しているシェルのプロセスのドメイン（`<kernel> /sbin/init /sbin/agetty /bin/login /bin/bash`）において，`/usr/bin/passwd`（パスワードの変更を行うコマンド）のアプリケーションの実行を許可する．この結果実行される `/usr/bin/passwd` のプロセスは，新たなドメイン（`<kernel> /sbin/init /sbin/agetty /bin/login /bin/bash /usr/bin/passwd`）において実行されるので，そのドメインに対して，`/usr/bin/passwd` が要求された処理を行うために必要なアクセス許可を定義しなければならない．

TOMOYO Linux のポリシーは，ドメインごとの定義ブロックが複数連続する構造を持ち，各ドメインのブロックは，ドメイン名の宣言で始まり，そのドメインに許可するアクセス許可を必要なだけ列挙したものとなる．

図 3 の 1 行目は，この後に続くアクセス許可定義の内容が，`/sbin/init` から実行された `/sbin/agetty` から実行された `/bin/login` から実行されたプログラム実行履歴を持つ `/bin/bash` に関するものであることを示している．これは，コンソールからログインしたシェルで `passwd` コマンドのアクセス許可定義を意味する．3 行目は，シンボリックリンクを解決した後のパス名が `/usr/bin/passwd` であるアプリケーションについて，呼び出し名（`exec.argv[0]`）が `passwd` であることを条件に，その実行を許可している．もし，特定の引数を指定した場合だけ実行を許可したい場合には，`exec.argv[]` として，すべての要素を条件として追加すれば良い．引数の個数を用いて制限したい場合には，`exec.argc=1` のように記述する．

TOMOYO Linux は，ホワイトリスト方式（陽に許可したもの以外はすべて拒否）を採用しているので，この定義ではコンソールからログインしたシェルから実行できるの

© 2012 Information Processing Society of Japan

```
 1  <kernel> /sbin/init /sbin/agetty /bin/login /bin/bash
 2
 3  file execute /usr/bin/passwd exec.realpath="/usr/bin/passwd" exec.argv[0]="passwd"
 4  file read/write /dev/tty
 5  file read /etc/passwd
 6  file read /etc/profile
 7  file read /home/harada/.bash_profile
 8  file read /home/harada/.bashrc
 9  file read /etc/bashrc
10  file write /dev/null
```

図 3 /bin/bash ドメインのポリシー（抜粋）
Fig. 3 Policy of /bin/bash Domain.

は，ここに書かれた条件を満たす/usr/bin/passwd のみで，他のアプリケーションを実行しようとするとエラーとなる．4 行目から 10 行目までは，このドメインに属する/bin/bash が読み書きできるファイルについて許可している．

図 4 は，図 3 で実行を許可された/usr/bin/passwd，すなわち「/sbin/init から実行された/sbin/agetty から実行された/bin/login から実行された/bin/bash から実行された」プログラム実行履歴を持つ/usr/bin/passwd についてのアクセス許可定義となる．passwd コマンドは，/etc/shadow の内容を読み込み，パスワードの内容を変更した結果を/etc/nshadow というファイルに書き込んでから，/etc/shadow にリネームを行っている．7 行目，9 行目，10 行目では，単に対象（客体）となるファイル名だけでなく，そのパーミッションやユーザ ID，グループ ID なども必須条件として指定されており，これらと合致しない場合にはアクセス要求は拒否される．ラベルに基づく MAC は，パス名やモードなどを扱わないので，このような条件を付与することはできない．

4.4.2 条件付きアクセス許可の例

条件付きアクセス許可について，必須条件のみのアクセス許可と任意条件を含むアクセス許可について，利用例を示す．

(i) **必須条件のみのアクセス許可の例**

必須条件のみアクセス許可の利用例について示す．

- file rename /etc/mtab.tmp /etc/mtab
  /etc/mtab.tmp というパス名を/etc/mtab というパス名にリネームすることを許可する．
- file create /var/lock/subsys/crond 0644
  /var/lock/subsys/crond の作成をパーミッション値が 0644 の場合のみ許可する．
- file chmod /dev/mem 0644
  /dev/mem のパーミッションについて 0644 にのみ設定することを許可する．
- file execute /bin/ls
  /bin/ls の実行を許可する．

(ii) **任意条件付きアクセス許可の例**

任意条件については，「属性=値」（属性が指定した値である場合は許可），あるいは「属性!=値」（属性が指定した値でない場合は許可）の形式で条件式を記述する．条件式は複数記述することが可能で，その場合すべての条件式が成立した場合のみアクセスが許可される．任意条件付きアクセス許可の利用例について示す．

- file symlink /dev/cdrom symlink.target="hdc"
  シンボリックリンクの内容が hdc の場合のみ，/dev/cdrom というシンボリックリンクの作成を許可する．
- file execute /bin/bash task.uid=500-1000
  ユーザ ID の範囲が 500 から 1000 の場合のみ/bin/bash の実行を許可する．
- file read /tmp/file001.tmp task.uid=path1.uid
  カレントプロセスのユーザ ID が/tmp/file001.tmp の所有者 ID と一致する場合だけそのファイルの参照を許可する．
- file execute /usr/bin/ssh exec.realpath="/usr/bin/ssh"
  exec.argv[0]="ssh"
  呼び出された名前が ssh で，シンボリックリンクを解決した結果のパス名が/usr/bin/ssh である場合に/usr/bin/ssh の実行を許可する．
- file execute /bin/bash exec.realpath="/bin/bash"
  exec.argv[0]="-bash"
  task.uid!=0 task.euid!=0
  呼び出された名前が-bash（ログインシェル）で，シンボリックリンクを解決した結果のパス名が/bin/bash で，かつプロセスのユーザ ID および実効ユーザ ID が 0（root ユーザ）ではない場合に/bin/bash の実行を許可する．

4.5 ポリシーの運用

本節では，TOMOYO Linux を用いたアクセス制御を行うために必要となるポリシーの策定および維持管理手順について述べる．

4.5.1 ポリシーの編集方法

システム起動時に，/etc/ccs/domain_policy.conf の

© 2012 Information Processing Society of Japan

```
 1  <kernel> /sbin/init /sbin/agetty /bin/login /bin/bash /usr/bin/passwd
 2
 3  file read /etc/passwd
 4  file read /etc/shadow
 5  file write /etc/.pwd.lock
 6  file read /dev/urandom
 7  file create /etc/nshadow 0666
 8  file write /etc/nshadow
 9  file chown/chgrp /etc/nshadow 0
10  file chmod /etc/nshadow 00
11  file rename /etc/nshadow /etc/shadow
```

図 4 /usr/bin/passwd ドメインのポリシー（抜粋）
Fig. 4 Policy of /usr/bin/passwd Domain.

内容が読み込まれる．このファイルは root ユーザを所有者とするファイルで，emacs などのテキストエディタを用いて編集することができる．このファイル自体へのアクセス要求も TOMOYO Linux のポリシーに従い保護されるので，「コンソールからログインして得られたシェルから実行された emacs のドメイン」のようにアクセスを許可するドメインを定義した上で，読み書き/参照のみなど操作のアクセス許可を必要に応じて条件を追加しながら定義する．

TOMOYO Linux では，システム起動後のポリシーの調整を行うためのツールとして，CUI (Character User Interface) のポリシーエディタが提供されている．図 2 はポリシーエディタの画面で，起動すると最初に現在のドメイン遷移が表示される．カーソル移動キーを用いてドメイン遷移の内容を確認し，リターンキーを押下すると，選択されていたドメインのアクセス許可の一覧が表示され，追加と削除を行うことができる．ポリシーエディタにおける操作は即時にアクセス制御の内容に反映される．ポリシー編集用のツールには他にポリシーロードとセーブを行うツールが提供されている．TOMOYO Linux の管理用に提供されているツールの内容と利用方法については，プロジェクトの Web ページ[*6]を参照されたい．

4.5.2 ポリシー策定の流れ

TOMOYO Linux のポリシー策定について，おおまかな流れを述べる．より詳しい内容については，プロジェクトが公開しているチュートリアル[*7]を参照されたい．

TOMOYO Linux には制御モードという概念がある．制御モードは，アクセス制御の動作内容を指定するもので，disabled, learning, permissive, enforcing の 4 種がある．各モードの意味について，**表 3** に示す．

モードは，ドメインごとに指定することができ，ポリシーファイルで定義するか，あるいはポリシーエディタで変更することができる．TOMOYO Linux のポリシー策定の基本的な流れは以下のようになる．

[*6] http://tomoyo.sourceforge.jp/1.8/man-pages/index.html
[*7] http://tomoyo.sourceforge.jp/about.html

表 3 TOMOYO Linux の制御モード
Table 3 TOMOYO Linux mode.

モード	内容	動作
disabled	無効	通常のカーネルと同様に動作する
learning	学習	ポリシー違反が発生しても要求を拒否しない ポリシー違反が発生しないようにするのに必要なアクセス許可をポリシーに追加する
permissive	確認	ポリシー違反が発生しても要求を拒否しない
enforcing	強制	ポリシー違反が発生したら要求を拒否する

(i) 一定期間 learning モードでシステムを稼働させる．これにより，プログラム実行履歴（ドメイン一覧）と各ドメインのアクセス要求の情報を収集する
(ii) 前項で得られた結果を確認し，ポリシー定義を作成する
(iii) permissive モードでシステムを稼働し，アクセス許可に不足がないか確認する
(iv) enforcing モードでシステムを稼働する

各工程に要する期間は，システムの内容と制限したい内容により変わる．enforcing モードで稼働するタイミング（ポリシーを凍結するタイミング）についての判断はシステム管理者が行わなければならない．提案方式を用いると，必要とするアプリケーションを必要とする状況（ドメイン）にて，実行することによって，実行履歴を得ることができる．その内容には，当該アプリケーションの中から呼び出されるアプリケーション群（およびさらにそれらから呼び出されるアプリケーション群）についても自動的に含まれるので，現システムのあらゆる状況で関連するすべてのアプリケーションを実行すれば，必要となる完全な実行履歴を得ることができる．したがって，これを実行可能な場合はそれでよい．しかし，一般にはこの組み合わせ数は膨大で，すべてを尽くすことは困難である．この問題は，一般に，システムのバグを取るために，あらゆるパターンの実行を行ってその結果をチェックすることが行われていることと同様である．したがって，本提案システムもそれと同様，実用上は次のような考え方で実行漏れを防ぐのが適切

© 2012 Information Processing Society of Japan

と考えており，実際そのような運用を行っている．

- できるだけ長い時間をかけてプログラム実行履歴の情報を収集する
  観測する期間を長くすることにより，アプリケーションの実行漏れが起こる可能性を減らすことができる．後述する Web サーバへの適用の実証実験では，約 2 週間かけて行っている．
- 評価プログラム，テストツールなどがあれば利用する
  アプリケーションごとに用意された評価プログラムやテストツールがあれば，それらを実行することによって，網羅的にアプリケーションを実行させることができる．
- 試験を徹底する
  ここでいう試験とは，ポリシーの試験ではなく，保護の対象とするコンピュータそのものであり，その上でサービスを提供するソフトウェアの試験のことである．
- バッチ処理，エラー処理について確認する
  上の項目すべてを行ったとしても，月次バッチや，ディザスタリカバリなど，常時実行されないアプリケーションについて，漏れが生じる可能性がある．バッチ処理については，システムの時間を変えることにより，確認することができ，ディザスタリカバリ等については，それが発動する状況を人為的に起こすことにより確認する．

内容が固まったら，確認用の機能ではなく，保護用の機能によりシステムを起動する．

4.5.3 ポリシーの更新

制御モードを enforcing に切り替えることにより，TOMOYO Linux による保護が有効となる．システム管理者は，ポリシーにないアクセス要求の有無と内容をログにより確認する．ログに残された要求の内容を確認した結果，該当する要求を許可すべきと判断したら，ポリシーエディタにより当該の内容を追加し，許可を与える．TOMOYO Linux のログの内容には，ドメイン名が含まれているため，そのエラーが生じた状況を特定することが可能で，それを反映すべき箇所も明確である．

OS のアップデート，アプリケーション（パッケージ）の追加や削除については，それによる影響が事前にわかる場合には，ポリシーエディタで修正，あるいは `/etc/ccs/domain_policy.conf` をテキストエディタで編集することにより対応することができる．そうでない場合については，一時的に MAC を無効にしてアップデートやインストール，削除を行ってからポリシーの修正を行う．これは，TOMOYO Linux に限らず MAC 共通の運用である．ここで MAC を無効にするのは，MAC では管理者権限に関係なくアクセス要求を判定するため，アップデート等が適用しないことが考えられるからである．

5. 評価

本章では，提案方式の評価として，その特長について整理し，使いやすさに関する初期検証の内容について述べる．また，提案方式を導入したことによる処理遅延について，TOMOYO Linux を用いて計測した結果について示す．

5.1 提案方式の特長

本研究の提案方式は以下の特長を持つ．

(i) **アプリケーションの処理内容を考慮したアクセス制御が行える**

前章の例で示したように，アプリケーションの処理内容とアクセスを許可した後のシステムの状態を考慮したアクセス制御を行うことができる．

(ii) **システムの動作内容を理解して設定できる**

MAC を導入したシステムでは，ポリシーにないアクセス要求がないかを監視し，それが生じた場合，ポリシー定義の漏れか，不正な要求が行われたかの切り分けを行い，その結果により対処を行う．提案方式では，アクセス要求がエラーとなった場合，そのプログラム実行履歴の情報が得られるので，状況の確認が容易である．

(iii) **アプリケーション（ドメイン）ごとに独立にアクセス許可を定義できる**

ラベルに基づく MAC の実装である SELinux では，情報システム全体を対象として，必要なラベルを定義する．あるラベルを持つアプリケーションでエラーが生じた場合，そのラベルを持つアプリケーション群と他のラベルとの遷移条件を考慮した上で対処を行うべきであるが，それは必ずしも容易ではない．4 章で示したポリシー例に見るように，本アクセス制御方式のポリシーはドメインごとに独立にして定義され，修正できるので管理は容易である．

(iv) **Role-Based Access Control としての使い方ができる**

本稿で提案したアクセス制御方式は，Role-Based Access Control Model[21] (RBAC) や Identity-Based Access Control Model 的な使い方も可能である．3 章のシェルのプログラム実行履歴の例を示したように，ログインシェルから `/bin/su` を実行して得られるシェルは，もとのシェルとは独立にアクセス許可を定義できる．`/bin/su` に限らずシェルをネストして起動しても同様であり，シェルの階層により与えるアクセス許可を変えることにより，シェルから実行されるアプリケーション単位で役割を持たせることも可能である．[22] ユーザ ID やグループ ID を条件として記述することにより，root（Linux/UNIX におけるシステム管理者アカウント）のみ，root 以外，特定のユーザ ID，グループ ID などアクセスを許可する条件を柔軟に記述することができる．

© 2012 Information Processing Society of Japan

5.2 使いやすさに関する初期検証

提案方式の運用について，ラベルに基づく MAC を採用している SELinux と机上での評価を行った．また，2007 年度，TOMOYO Linux について行った実証実験の結果について紹介する．

5.2.1 運用面での効果

独立行政法人情報処理推進機構では，被験者を用いた調査の結果に基づき，SELinux を用いた「セキュアな Web サーバ構築のガイドライン」[23] を公開している．同ガイドラインでは，SELinux の適用について，対象アプリケーションの仕様調査と「標準セキュリティポリシー[*8]定義の調査・解析・修正あるいは新規セキュリティポリシーの定義について，大きな作業工数が必要となると結論づけている．標準セキュリティポリシーが提供されていることは，システム管理者にとって福音であるが，標準セキュリティポリシーが想定する環境とシステム管理者が管理するシステムの環境は一致しない．標準セキュリティポリシーの内容を管理対象のシステムに合わせるために，あるいはエラーが生じたときにその対処を検討する際に，システム管理者は膨大なポリシーの内容を理解しなければならなくなる．

提案方式では，アプリケーションの実行状況の分類はカーネル内で自動的に行われ，また分類間の関係も容易に理解できるので，システム管理者はそれぞれの実行状況（ドメイン）ごとのアクセス許可定義に専念することができる．

5.2.2 実サーバへの導入評価

2007 年度，NPO 日本ネットワークセキュリティ協会のセキュア OS 普及促進 WG（当時）では，セキュア OS 導入の内容，効果，システム管理面の影響などを評価することを目的に，同協会の Web サーバに TOMOYO Linux を導入する実証実験を行った．実験では，インターネットに公開されている Apache Web サーバを対象として，ファイルに関するアクセス制御を設定した．Web サーバでは CGI を使用していたが，CGI は Apache とは別のドメインとして管理され，ログにより収集されたファイルアクセスの要求の履歴から，それぞれのドメインに必要なアクセス許可をファイル単位で定義している．実験参加者は，TOMOYO Linux の動作原理，ポリシーの編集については，ごく短時間で理解することができ，ポリシーの策定の作業を通じて，Linux サーバの動作内容について把握することができたという感想がある．これらの効果は，ポリシーでパス名をそのまま記述できること，プログラム実行履歴によるアクセス制御の分類が管理者にとって理解しやすいことによると思われる．実験参加メンバーによる報告書[*9] [*10]が公開されている．

[*8] 前述のディストリビューションの標準設定に合わせたポリシーのこと．

5.3 性能への影響

本節では，TOMOYO Linux を用いて，提案方式による性能への影響の測定結果について，システムコール種別による影響とポリシーの規模による影響について計測を行った結果について示し，考察を行う．

5.3.1 システムコール種別による影響

UNIX 系システム用のベンチマークツール, LMBench[24] を用いてシステムコール種別による影響について計測を行った．

提案方式を適用した場合，性能による影響は原理的にはフックされるシステムコールが対象となり，フックされないシステムコールは影響を受けないことが予想される．そこで，LMBench で計測できる OS 関連のシステムコールのうち，TOMOYO Linux がフックしないものと，フックするものについて分けて比較する．比較の条件については，TOMOYO Linux をインストールしていない状態と，TOMOYO Linux によるアクセス制御を有効にした状態とで，それぞれ LMBench を実行した．

LMBench の実行環境について**表 4** に示す．LMBench でできる計測項目および詳細な仕様については，同ツールの Web ページ[*11]に掲載されているので，ここでは説明しない．TOMOYO Linux がフックしていないシステムコールを対象にした計測結果を**表 5** に，TOMOYO Linux がフックしているシステムコールを対象にした計測結果について，**表 6** に示す．なお，ベンチマークの実施にあたっては，ベンチマークの実行に必要なプロセス以外が実行されていないようにした上で行った．表 5, 表 6 の読み方について説明する．列，Func. は LMBench のテスト項目，列，Base は TOMOYO Linux をインストールしていない状態の計測結果 (μsec)，列，TOMOYO は TOMOYO Linux を導入し，MAC を有効にした状態の計測結果 (μsec)，列，Diff は TOMOYO と Base の差分 (μsec)，列，Overhead は，

$$Overhead = \frac{TOMOYO - Base}{Base} \times 100$$

により得られた数値であり, Overhead の値が 100 であれば，TOMOYO Linux を有効にしない場合に対して，100%の遅延が生じたこと，すなわち処理に要する時間が 2 倍になったことを意味する．

表 5 を見ると，TOMOYO Linux を有効にした場合の遅延の割合は，±5%の範囲に収まっている．理論上，TOMOYO Linux を有効にすることにより処理速度が向上することはないため，これらは，LMBench による計測の誤

[*9] http://www.jnsa.org/result/2007/tech/secos/

[*10] ポリシーの抜粋も収録されているがバージョンの相違により本稿の例とは異なっている．

[*11] http://www.bitmover.com/lmbench/

© 2012 Information Processing Society of Japan

表 5 LMBench の結果（フック対象外のシステムコール分）
Table 5 Result of LMBench (not hooked).

Func.	Base (μsec)	TOMOYO (μsec)	Diff (μsec)	Overhead (%)
null syscall	0.274	0.269	0.0	-1.82
null I/O	0.4365	0.418	0.0	-4.24
Select on 100 tcp fd's	7.0815	7.1455	0.1	0.90
Signal handler installation	0.552	0.56	0.0	1.45
2p/0K ctxsw	10.97	10.665	-0.3	-2.78
2p/16K ctxsw	11.26	11.07	-0.2	-1.69
2p/64K ctxsw	14.21	14.39	0.2	1.27
8p/16K ctxsw	12.22	11.755	-0.5	-3.81
8p/64K ctxsw	14.035	14.095	0.1	0.43
16p/16K ctxsw	12.185	12.04	-0.1	-1.19
16p/64K ctxsw	14.135	14.225	0.1	0.64
Pipe	38.3	36.73	-1.6	-4.10
AF UNIX	24.47	24.28	-0.2	-0.78
Mmap	2341.55	2375.1	33.5	1.43
Page Fault	2.52829	2.58089	0.1	2.08
Select on 100 fd's	3.254	3.35045	0.1	2.96

表 6 LMBench の結果（フック対象のシステムコール分）
Table 6 Result of LMBench (hooked by TOMOYO).

Func.	Base (μsec)	TOMOYO (μsec)	Diff (μsec)	Overhead (%)
Simple stat	3.12	7.1145	4.0	128.03
Simple open/close	5.037	9.5065	4.5	88.73
Signal handler overhead	3.8015	5.961	2.2	56.81
Process fork+exit	300.35	301.7	1.3	0.45
Process fork+execve	1001.95	1062.55	60.6	6.05
Process fork+/bin/sh -c	2226.1	2551.2	325.1	14.60
UDP	54.65	69.91	15.3	27.92
RPC/UDP	61.565	80.615	19.1	30.94
TCP	58.04	57.3	-0.7	-1.27
RPC/TCP	72.64	71.36	-1.3	-1.76
TCP/IP connection cost	65.8	72.4	6.6	10.03
0K File Create	25.32	41.035	15.7	62.07
0K File Delete	19.865	27.795	7.9	39.92
10K File Create	80.9	95.855	15.0	18.49
10K File Delete	41.075	50.33	9.3	22.53

© 2012 Information Processing Society of Japan

表 4 ベンチマークに用いた環境
Table 4 Benchmark Envrionment.

	specification/version
CPU	Core 2 Duo T7200 2.0GHz
Memory	2GB
OS	Ubuntu 10.04 x86_64
Kernel	2.6.32-39.86
TOMOYO Linux	1.8.3p5
Benchmark tool	LMBench 3.0-a9

差であると考えられる．

表 6 を見ると，stat, open/close, signal handler について，50%以上の遅延が生じている．0K File Create（空のファイルの作成）について，60%以上の遅延となっているが，10K File Create については，18.49%とかえって値が小さくなっている．後者は空のファイルを作成した後で，10KB のデータの write を行っており，TOMOYO ではフックしていない処理が加わることによって，遅延時間の相対的な割合が小さくなった結果このように見えると考えられる．

プログラム実行履歴の処理の影響を受けると考えられる fork 関連の項目については，fork と exec の組み合わせで 5%程度の遅延が生じている．fork+/bin/sh -c について，値が大きくなっているのは，/bin/sh の exec が行われることにより，exec の処理が 2 度実行されるためと考えられる．

LSM を用いて，MAC（セキュア OS）の性能評価を行う試みとして，LSM Performance Monitor (LSMPMON)[25] がある．LSMPMON を用いるとフック関数の呼び出し回数と処理時間を記録できるが，ユーザやシステム管理者が体感する性能への影響の予測という点では，課題が解決されておらず，今後の研究が期待される．

5.3.2 ポリシーの規模による影響

パッケージのインストールなどにより，実行されるアプリケーションの数が増加し，ポリシー定義が増大することによる性能への影響について評価を行った．

提案方式では，アプリケーションの実行（execve システムコール実行時）にそのプロセスに対応するドメインを割り当てる．次に，要求された内容が，当該ドメイン内のアクセス許可の内容に含まれているかどうかを判定を行う．実行されるアプリケーション数が増加し，ドメイン数が増加することにより，アクセス要求ごとに該当するドメインを探すための処理時間が増加する．また，アプリケーションの処理内容が複雑化することにより，ドメインごとのアクセス許可数が増加し，該当するアクセス許可を探すための処理時間が増加する．したがって，ポリシーの規模が増大することによる性能への影響について，次の 2 つにより影響を計測することができる．

- ドメイン増加によるドメイン探索時間の増大
- アクセス許可数増加によるアクセス許可探索時間の増大

ドメイン探索時間の増大については，指定した数値だけドメインを機械的に生成するプログラムを作成し，ドメイン数が 2 から 100000 までの範囲について/tmp/reexec（自分自身を指定された回数，実行するプログラム）を実行した場合の処理時間について計測した．各ケースについて，/tmp/reexec のためのドメインは最後に見つかるようにしている．ドメイン数と処理時間の関係について，**図 5** に示す（ドメイン数は対数軸を用いている）．

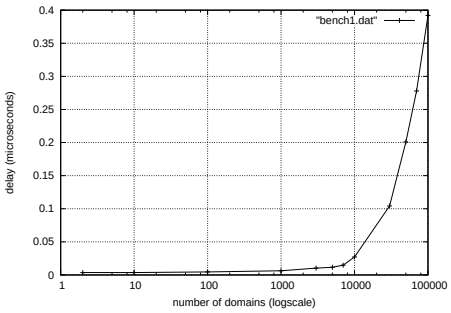


図 5 ドメイン数増加による処理速度の遅延
Fig. 5 Performace delay due to domain number increase.

ドメイン数が 2 の場合について，TOMOYO Linux が有効な場合と，TOMOYO Linux がインストールされていない場合の差は，0.00362μsec だった．図 5 では，ドメイン数について対数軸を用いているが，予想どおりドメイン数の増加に伴いドメイン探索に要する時間の遅延が増大し，ドメイン数が 10000 を超えると増加の割合が顕著となっていることがわかる．

アクセス許可探索時間の遅延については，指定した数値だけアクセス許可を機械的に生成するプログラムを作成し，1 から 100000 までの範囲について，/dev/null を 10000 回 open する時間を計測した．各ケースについて/dev/null の open を許可するエントリは最後に見つかるようにしている．アクセス許可の行数と処理時間の関係について，**図 6** に示す．アクセス許可の行数は対数軸を用いている．アクセス許可数が 1 の場合について，TOMOYO Linux が有効な場合と，TOMOYO Linux がインストールされていない場合の差は，0.0032μsec だった．

図 6 では，アクセス許可数について対数軸を用いているが，予想どおりアクセス許可数の増加に伴いアクセス許可の探索に要する時間が増大し，アクセス許可数が 10000 を超えると増加の割合が顕著となっていることがわかる．

2005 年に最初のバージョンを公開してから現在までの経験則として，主要な Linux ディストリビューションに TOMOYO Linux を導入して利用する場合，ドメイン数およびドメイン内のアクセス許可数について，それぞれ 2000 未満の範囲に収まることが確認されている．パッケージの

© 2012 Information Processing Society of Japan

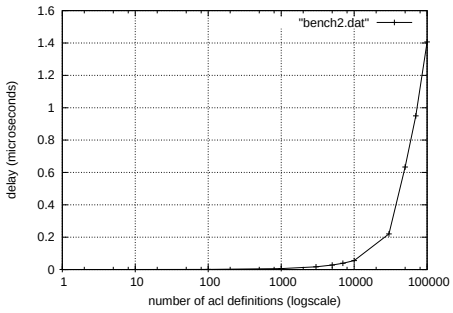


図 6 ドメイン内アクセス許可数増加による処理速度の遅延
Fig. 6 Performace delay due to ACL number increase.

追加を行い，実行するアプリケーションの数が増えることによって，ドメイン数は増加するが，図 5 および図 6 の結果を見ると，通常の利用においては，性能上大きな支障は生じないと考えられる．

6. 考察

本章では，提案方式に対する考察として，関連性の高い他の方式との比較を行い，典型的な不正アクセス手法に対する効果について確認する．さらに，提案方式では解決できないケースと，その対策として，機能を強化する際の課題について述べる．

6.1 他の方式との比較

6.1.1 ラベルに基づく MAC との比較

(i) ラベルに基づく MAC の特長と適した用途

ラベルに基づく MAC では，主体と客体について，リネームなど名前の変更に左右されることなく，アクセス可否判断を行うことができる，また，名前を用いないので，名前を持たない資源にも適用することができる．これらのことから，ラベルに基づく MAC は，主体から客体へのアクセス可否を堅固に制限するという用途に適したものと言える．

(ii) 提案方式の特長と適した用途

提案方式では，アプリケーションの処理内容に影響を与える要因をアクセス可否判断のパラメータとして活用することにより，アクセスを許可する場合を絞り込んでいくことができる．パラメータとして，ファイル名や DAC のパーミッションなどを用いることができるので，それらの変更をシステム管理者が認めた範囲内に収めることができる．このことから，提案方式は，情報システムの状態を正しく維持し，余計な機能を利用させないという用途に適したものと言える．

(iii) 提案方式の位置付け

どのような方式が適しているかは，どのような脅威を防ぎたいかにより決まる．TCSEC が発表された 1983 年の時点では，機密情報の漏えいが主な関心事であり，防ぎたい脅威であった．資格を持つアプリケーションだけが，それらが必要なファイルにアクセスできること，およびファイルから得られた情報が漏えいしないことが重要であった．ラベルに基づく MAC は，このニーズを満たしているが，アプリケーションの処理の内容には関心が払われていなかった．提案方式は，個々のアプリケーションのプロセスが行う処理の内容に注目し，必要とする処理以外を行わせないことを目指している．提案方式は，ラベルに基づくアクセス制御方式の代替ではなく，その限界を補完するものと位置づけることができる．

6.1.2 AppArmor との比較

TOMOYO Linux は 2005 年 11 月に，AppArmor は 2006 年 1 月に公開された．公開されるまでお互いにパス名に基づく MAC を開発していることを知らなかったため，同じパス名方式でありながら異なる実装方式となっている．

(i) 実装面での違い

TOMOYO Linux は，AppArmor よりも先に，様々な要因（コマンドライン引数や環境変数などの条件パラメータ）を考慮したアクセス可否判断の必要性に気付き，実際に実装を行い，既にメインラインに採用されている．AppArmor でも，どのような条件（パラメータ）をチェックすべきかの議論が始まり，2011 年の 11 月，AppArmor 開発者のメーリングリストにアクセス制御の際に環境変数の値を用いることが提案[*12]された．メーリングリストの議論では，TOMOYO Linux における仕様や実装方法について紹介され，今後 AppArmor においても採用される可能性がある．

(ii) 提案方式の適用可能性

AppArmor と TOMOYO Linux は，客体についてパス名を用いる点において共通しているが，主体の分類について実装方式が大きく異なり，同一の方式と見なすことは適当ではない．具体的には，TOMOYO Linux では，起動時からのすべての履歴を記録してその内容に基づいてポリシーを定義できるので，起動時からシャットダウンまでのすべてのプロセスを対象としたアクセス制限が実現可能である．AppArmor では，起動時からの履歴は記憶していないので，デーモンや Web ブラウザなど特定のプロセスを対象としたアクセス制御しか実現できない．これは，設計思想の違いに由来する差異である．

(iii) AppArmor にない TOMOYO Linux の工夫点

AppArmor にない TOMOYO Linux の工夫点について，以下に示す．

- システムの起動時からシャットダウンまでのすべてのプロセスを対象としたアクセス制限が可能である．
- ドメイン遷移がツリー構造になることを応用することで，RBAC 的な使い方やログイン認証の強化も可能である．
- ワイルドカードとして解釈する文字を \（バックスラッシュ）でエスケープするという方式であるため，互換性を維持しながらワイルドカードの種類を増やすことがで

[*12] https://lists.ubuntu.com/archives/apparmor/2011-November/001668.html

© 2012 Information Processing Society of Japan

きる．

- .git のように特定の意味を持つディレクトリ名やファイル名を除外するために，\-という減算演算子を備えている．
- できあいのポリシーを用いないため，対象となるシステム向けに無駄（不要なアクセス許可）の無いポリシーを定義することができる．そして，自分で内容を理解して設定しているため，問題が起きたときでも無効にしてしまうことが無い．

6.1.3 Context-aware Access Control 取り組みとの比較

近年, コンテキストを反映したアクセス制御 (CAAC: Context-aware Access Control) に関する取り組みが活発に行われている．

Matthias Baldauf らの “A survey on context-aware systems”[26] では，Context-aware system について，「ユーザの積極的な介在を必要とせずに環境的なコンテキストを考慮することにより，ユーザビリティや効率を向上することを目指すもの」と定義している．今日 CAAC に関連する取り組みとしては，モバイルやユビキタスの領域で，位置情報やネットワーク情報を context と扱うものおよび Web サービスに context の概念を導入する試みなどが多い [27]．CAAC における context は抽象的な概念であり，利用できる情報を集めて活用することにより付加価値を目指すと理解すれば，本論文で提案方式も CAAC に関連すると分類できる．

CAAC に関する取り組みで，本論文の提案に近いものとして，Salvia[28] がある．Salvia では，コンテキストを内部と外部の 2 種に分けて，内部コンテキストとしては OS 内部で取得可能なプロセスの属性値とシステムコールの履歴を，外部コンテキストとしては位置情報，絶対時刻，無線 LAN アクセスポイント (ESSID) などを用いるプロトタイプを構築している．内部コンテキストとしてプロセスの属性値を利用するという点については，本論文におけるアプリケーションの処理内容の判定の考え方に，システムコールの履歴はプロセスの状態の細分化を目指しているという点で，目指しているところは本論文の提案に近いと言える．しかし，システムコールについてシステムコール番号，引数，返り値をもってそのプロセスが実行される状況を推測するには，詳細な分析が必要となる．特定の場合のみを抽出することは可能でも，全面的に利用することは困難と考えられる．提案方式では，内部的コンテキストとしてドメインを用いることで，より簡便に実現できる利点がある．

6.2 不正アクセス手法に対する効果

提案方式について，典型的な不正アクセス（攻撃）の手法 [29] への効果について評価する．

(i) **不正なアプリケーションの実行**

ここで不正とは，システム管理者が望まないアプリケーションの実行を指す．典型的な例としては，ネットワークを介してサービスを提供しているサーバの脆弱性を突いてこれを乗っ取り，シェルを実行する場合（およびそのシェルから実行するアプリケーション）がこれにあたる．バッファオーバーフロー攻撃，書式文字列攻撃，OS コマンドインジェクション攻撃なども同様である．バッファオーバーフロー攻撃では，シェルを起動してから不正操作を行うのが一般的である．対象となるアプリケーションについて，/bin/sh の実行を許可しなければ，脆弱性を持っているアプリケーションが奪われたとしてもシェルの起動に失敗し，それ以降の操作を許さないようにすることができる．ラベルに基づく MAC でこれを行うには，主体の状況とその遷移，各状況に置かれた主体に対する客体の組み合わせにより主体および客体のラベル定義が必要となる．本研究の提案方式では，各プロセスについてその置かれた状況を示すプログラム実行履歴を持つため，管理者が新たにそれを定義する必要はない．また，本来の処理としてシェルの実行が必要となる場合には，方式を問わずその実行を許可しなければならないが，提案方式を用いればコマンドライン引数，環境変数などをアプリケーション実行の制約条件として限定することができるので，悪用されるリスクを軽減できる．

(ii) **パス・トラバーサル攻撃**

相対パスなどを利用し，本来アプリケーションがアクセスする必要のないファイルの読み書きを行う攻撃がある．本稿で提案する方式について Linux 上に実装する場合には，プログラム実行履歴やパス名について，それらのカーネル内におけるデータ講造に変換したものから絶対パス名を逆算し，そのパス名を用いていることによって，パス・トラバーサル攻撃を無効とすることができる．

(iii) **シンボリックリンク攻撃**

シンボリックリンクは，対象となるファイル（パス名）に対して，異なるファイルであるかのように見せる効果を持つ．これを悪用することにより，アプリケーションにリンク先のファイルを破壊させる攻撃をシンボリック攻撃と呼ぶ．本稿で提案する方式を用いる場合，まず攻撃に用いられるディレクトリにシンボリックの作成が許可されていなければ攻撃が成立しない．もし，許可されていた場合でもさらに作成しようとするシンボリックのリンク先（ファイル名やディレクトリ名の文字列）を制約条件として限定することができるので，悪用されるリスクを軽減できる．

6.3 機能強化に向けた課題

提案方式は，アプリケーション実行要求（execve()）を契機として，アクセス主体ごとのプログラム実行履歴を導出し，主体の実行状況を分類する．そのため，execve() を伴わない限り，アクセス主体の実行状況を分離することが

© 2012 Information Processing Society of Japan

できない問題がある．

例えば，Web サーバの Apache は，CGI (Common Gateway Interface) をサポートしており，一般的に利用されているが，提案方式では，CGI の実行方法が `execve()` を伴う場合には，その CGI は独立したドメインとなるが，`mod_perl` のように `execve()` を伴わない場合には，Apache と同じドメインになってしまう．これについては，アプリケーションが，新たにドメインを分けたいという情報を伝える仕組みを実現すれば，`execve()` を伴わない CGI についても，独立なドメインとして扱うことができるようになる．

提案方式は，アプリケーションの修正を必要とせずに，任意のアプリケーションに適用できる利点を持つが，アプリケーション自体の申告を併用することにより，アプリケーションの作者がより直接，具体的に不要なアクセス要求を排除することが可能となる．この申告方式を実現することが，今後の課題である．

また，別の問題としては，例えば，共有ディレクトリに置かれた一時ファイルについて，それらを作成したアプリケーションに限定してアクセスを許可するような場合には，それぞれのファイルについて，その所有者に関する情報を覚えさせておき，それに基づいたアクセス制御を行う必要がある．そのような場合は，提案方式をラベルに基づく MAC と併用することにより課題を実現することができる．

その他にも，仮想化されたサーバを実行するアプリケーションについて，同じプログラム実行履歴を持つアプリケーションは，実行している内容が異なる仮想マシンであってもアクセス制御の許可内容を区別することができない．仮想マシンごとにアクセスできる範囲を制限し，互いに影響を及ぼさないようそれらのケースへ適用する方法も今後の課題である．

7. おわりに

本稿では，従来のアクセス制御では考慮されていなかったアプリケーションの処理内容に注目するアクセス制御方式を提案し，それを実現するための方法について，筆者らが Linux 上に実装した TOMOYO Linux を例にあげて示した．提案方式では，アプリケーションの実行状況ごとに，各アプリケーションが行おうとする処理を考慮したアクセス制御を可能とする．それにより，従来のアクセス制御方式では識別できなかった不正，不要なアクセス要求を拒否し，情報セキュリティを高めることができる．提案方式の有効性と応用可能性について，TOMOYO Linux ポリシーの例により示し，提案方式を実装した情報システムの使いやすさについて，ラベルに基づく MAC との机上比較，および TOMOYO Linux の実証実験の結果について紹介した．また，典型的な不正アクセス手法に対する効果について考察を行い，提案方式が典型的な攻撃手法に対して有効であり，方式に起因する重要な弱点がないことを確認した．性能への影響の度合いについて，TOMOYO Linux を用いた計測結果を用いて，影響の内容と程度についての考察を示した．今後は，アプリケーションからの申告の併用による，アプリケーション実行状況の精度の向上や，使いやすさに関する検証方法の検討などを行う予定である．

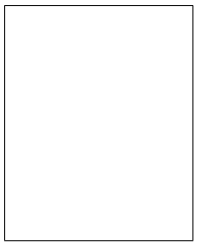
謝辞 査読者の方々より多数の貴重なご助言をいただきました．感謝いたします．本研究の論文化について，情報セキュリティ大学院大学，板倉征男教授（当時．後に同大学名誉教授．2011 年 8 月ご逝去）から懇切なるご指導をいただきました．板倉先生にご指導いただくことがなければ，本論文が世に出ることはありませんでした．ここにあらためて先生への感謝を記し，心からご冥福をお祈りいたします．

参考文献

[1] 原田季栄，半田哲夫，板倉征男：TOMOYO Linux の設計と実装，コンピュータシステム・シンポジウム論文集，pp. 101–110 (2009).
[2] 原田季栄，半田哲夫：Linux のセキュリティ機能：4．ラベルに基づくセキュリティの限界とその補完 TOMOYO Linux の設計思想と試み，情報処理， Vol. 51, No. 10, pp. 1276–1283 (2010).
[3] Peterson, D. S., Bishop, M. and Pandey, R.: Flexible Containment Mechanism for Executing Untrsted Code, *11th USENIX Security Symposium*, pp. 207–225 (2002).
[4] 大山恵弘：ネイティブコードのためのサンドボックスの技術，コンピュータソフトウェア，Vol. 20, No. 4, pp. 55–72 (2003).
[5] Goldberg, I., Wagner, D., Thomas, R. and Brewer, E.: A secure environment for untrusted helper applications confining the Wily Hacker, *Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography-Volume 6*, USENIX Association, pp. 1–1 (1996).
[6] Barth, A., Jackson, C., Reis, C. and Team, T.: The security architecture of the Chromium browser (2008).
[7] Loscocco, P. A., Smalley, S. D., Muckerbauer, P. A., Taylor, R. C., Turner, S. J. and Farrell, J. F.: The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments, *21st National Information Systems Security Conference*, Vol. 10, No. 2, pp. 303–314 (1989).
[8] Bishop, M.: Computer Security: Art and Science. 2003.
[9] Tcsec, D.: Trusted computer system evaluation criteria, *DoD 5200.28-STD*, Vol. 83 (1983).
[10] Peter Loscocco, N.: Integrating flexible support for security policies into the Linux operating system, *Proceedings of the FREENIX Track 2001 USENIX annual technical conference, June 25-30, 2001, Boston, Massachusetts, USA*, Citeseer, p. 29 (2001).
[11] Loscocco, P. A. and Smalley, S. D.: Meeting Critical Security Objectives with Security-Enhanced Linux, *Ottawa Linux Symposium* (2001).
[12] Smalley, S.: Configuring the SELinux policy, *NAI Laboratories* (2005).
[13] Schaufler, C.: Smack in embedded computing, *Proceedings of the 10th Linux Symposium* (2008).
[14] Cowan, C., Beattie, S., Kroah-Hartman, G., Pu, C., Wagle, P. and Gligor, V.: Subdomain: Parsimonious server

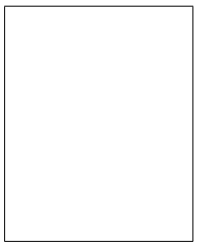
© 2012 Information Processing Society of Japan

security, *Proceedings of the 14th USENIX conference on System administration*, USENIX Association, pp. 355–368 (2000).
[15] Winter, J.: Trusted computing building blocks for embedded linux-based ARM trustzone platforms, *Proceedings of the 3rd ACM workshop on Scalable trusted computing*, ACM, pp. 21–30 (2008).
[16] Watson, R., Anderson, J., Laurie, B. and Kennaway, K.: Capsicum: practical capabilities for UNIX, *USENIX Security* (2010).
[17] Ken, W.: Buffer Overflow Attacks and Their Countermeasures., コンピュータソフトウェア, Vol. 19, No. 1, pp. 49–63 (online), available from ⟨http://ci.nii.ac.jp/naid/110003744115/en/⟩ (2002-01-15).
[18] Sandhu, R. and Samarati, P.: Access control: principle and practice, *Communications Magazine, IEEE*, Vol. 32, No. 9, pp. 40–48 (1994).
[19] 榮樂恒太郎, 新城 靖, 板野肯三：システム・コールに対するラッパ/リファレンス・モニター SysGuard の設計と実現, 情報処理学会論文誌, Vol. 43, No. 6, pp. 1690–1701 (2002).
[20] Wright, C., Cowan, C., Smalley, S., Morris, J. and Kroah-Hartman, G.: Linux security modules: General security support for the Linux kernel (2003).
[21] Sandhu, R., Coyne, E., Feinstein, H. and Youman, C.: Role-based access control models, *Computer*, Vol. 29, No. 2, pp. 38–47 (1996).
[22] 原田季栄, 松本隆明：セキュリティ強化 OS によるログイン認証の強化手法, 静岡大学情報学研究, Vol. 11, pp. 93–102（オンライン）, 入手先 ⟨http://ci.nii.ac.jp/naid/110007117435/⟩ (2005).
[23] 情報処理振興技術協会セキュリティセンター：ガイドライン「セキュアなインターネットサーバの設定と運用」, 技術報告 (2003).
[24] McVoy, L. and Staelin, C.: lmbench: Portable tools for performance analysis, *Proceedings of the 1996 annual conference on USENIX Annual Technical Conference*, Usenix Association, pp. 23–23 (1996).
[25] 松田直人, 佐藤和哉, 田端利宏, 宗藤誠治：LSM を利用したセキュア OS の性能評価機能の実現と評価, 電子情報通信学会論文誌 D, Vol. J92-D, No. 7, pp. 963–974 (2009).
[26] Baldauf, M., Dustdar, S. and Rosenberg, F.: A survey on context-aware systems, *International Journal of Ad Hoc and Ubiquitous Computing*, Vol. 2, No. 4, pp. 263–277 (2007).
[27] Truong, H. and Dustdar, S.: A survey on context-aware web service systems, *International Journal of Web Information Systems*, Vol. 5, No. 1, pp. 5–31 (2009).
[28] KAZUHISA, S., YOSHIMI, I., KOICHI, M. and EIJI, O.: An Adaptive Data Protection Method based on Contexts of Data Access in Privacy-Aware Operating System Salvia(Operating System), 情報処理学会論文誌. コンピューティングシステム, Vol. 47, No. 3, pp. 1–15 (online), available from ⟨http://ci.nii.ac.jp/naid/110004708857/en/⟩ (2006-03-15).
[29] 品川高廣：オペレーティングシステムによる不正アクセス防止技術, コンピュータソフトウェア, Vol. 21, No. 6, pp. 482–493 (2004).



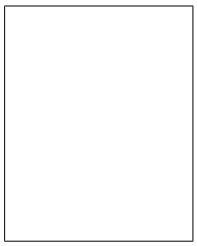
原田 季栄 （正会員）

1985 年, 北海道大学工学部応用物理学科工業数学講座卒業. 同年日本電信電話（株）入社. 2003 年より（株）NTT データにて, オープンソースのプロジェクトマネージメントに携わり, オペレーティングシステムのセキュリティ強化に興味を持つ. 2012 年, 情報セキュリティ大学院大学情報セキュリティ研究科修了, 博士（情報学）. 情報処理学会, 電子情報通信学会, IEEE, ACM 各会員.



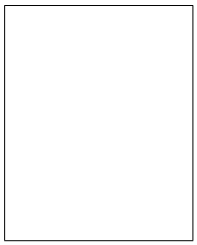
半田 哲夫

2001 年青山学院大学理工学部電気電子工学科卒業. 同年 NTT データカスタマサービス（株）入社. 現在 NTT データ先端技術（株）に勤務. 2003 年より Linux カーネル開発に従事, オペレーティングシステムのセキュリティ強化に興味を持つ.



橋本 正樹 （正会員）

2010 年 3 月, 情報セキュリティ大学院大学情報セキュリティ研究科修了, 博士（情報学）. 同年 4 月より情報セキュリティ大学院大学情報セキュリティ研究科・助教. 情報処理学会, 電子情報通信学会, 日本ソフトウェア科学会, IEEE 各会員. 電子情報通信学会 ISS・情報通信システムセキュリティ研究会専門委員.



田中 英彦 （正会員）

昭和 45 年東京大学大学院博士課程修了, 工学博士. 東京大学工学部教授、同情報理工学系研究科教授・研究科長を経て、平成 16 年情報セキュリティ大学院大学教授, 研究科長. 平成 24 年同大学学長. 計算機アーキテクチャ, 分散処理, 知識処理, デペンダブル情報システム等に興味を持つ. 著書に「非ノイマンコンピュータ」「計算機アーキテクチャ」「Parallel Inference Engine」などがある. 情報処理学会名誉員、電子情報通信学会, 人工知能学会, 各フェロー, IEEE ライフフェロー.

© 2012 Information Processing Society of Japan