社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－DPS－122（31）
2005－CSEC－28（31）
2005／3／22

# 細粒度権限管理実現のための一手法の提案

稲村 雄† 竹下 敦†

†株式会社 NTT ドコモ マルチメディア研究所
〒 239-8536 神奈川県横須賀市光の丘 3-5
NTT DoCoMo R&D センタ

{jane,takeshita}@mml.yrp.nttdocomo.co.jp

従来型 OS におけるユーザ/グループ ID 等に基づく粗粒度な権限管理の枠組によってもたらされる限界を打破するための一手法として，動的なポリシー変更が困難である SELinux 等の必須アクセス制御 (MAC) をベースとする OS とは異なり，細かいポリシー変更可能かつユーザビリティを損なわない暗号技術を用いた細粒度権限管理方式を提案する．ポリシー

# A method for obtaining fine-grain privilege control mechanisms

Yu Inamura† Atsushi Takeshita†

†NTT DoCoMo, Inc. Multimedia Laboratories
3-5, Hikarinooka, Yokosuka
Kanagawa, 239-8536 Japan

{jane,takeshita}@mml.yrp.nttdocomo.co.jp

**Abstract** We propose a method, which is based on cryptography techniques, for obtaining a fine-grain privilege control method. Unlike the OSes based on mandatory access control (MAC) such as the SELinux, which require cumbersome maintenance cost for management, the proposed method is quite usable and can easily be integrated into the existing OSes.

## 1 はじめに

一般的な UNIX や Windows のような任意アクセス制御[1]ベースの OS では，root や Administrator 等の管理者権限を持つユーザはシステムに対して事実上任意の操作が可能であるため，万一 root 権限下で動作するプロセスが攻撃者に乗っ取られた場合に甚大な被害を及ぼすという問題がある．

このような危険性を排除するための手法として，必須アクセス制御[2]なる権限管理技術の提案がなされている．SELinux[1] に代表される MAC ベースの OS では，従来型 UNIX 系 OS におけるユーザ/グループ ID に基づくアクセス制御機構[3]に加えて，役割に基づくより細粒度の権限管理を必須とすることで，たとえ root 権限で動作するプロセスが徴発されたとしてもシステムに対して大きな損害を与えることはできないという環境を実現している．

本稿ではそのような MAC ベースの OS の代わりとなる，もしくは補完することを目的として，動的なポリシー変更が容易な暗号技術に基づく新しい細粒度権限管理手法を提案する．

以降の構成は以下の通りである．まず，2 章で MAC ベース OS に存在する問題点を述べた後，3 章で提案方式の概要を示す．4 章で同方式の試験実装結果を報告し，5 章で全体のまとめおよび今後の課題等の提示を行なう．

[1]Discrete Access Control (DAC)
[2]Mandatory Access Control (MAC)
[3]ファイルに対する読出/書込/実行権制御等

## 2 MACベースOSの問題点

SELinux等のMACに基づくOSでは，システムに存在するすべてのオブジェクト[4]に対して厳格かつ細分化されたアクセス権限が設定され，特定のオブジェクトにはOSによって明に許可された主体(サブジェクト)のみがアクセス可能となるように図られる．

具体例としてSELinuxでは，同じ性格を持つオブジェクトの集合に対して“型”(タイプ)[5]を割り当て，操作主体となるサブジェクトを“役割”[6]としてまとめ，どの役割を持ったサブジェクトがどの型のオブジェクトに対してどのような操作が可能かという制限を，基本的には静的なポリシー・データとして設定し，オブジェクトへのアクセス時に該ポリシー・データを参照して可否を判定するという形で権限管理が実現される．

このような方式では，システム全体で一つの統一したポリシーを維持する必要があるため，動的な変更が行ない難いという点が大きな問題と考える．たとえばSELinuxではポリシーを動的に変更する仕組は用意されているものの，その仕組を用いて実際に変更を行なうには，

- 特定ディレクトリ[7]下に置かれるファイル群に記述されるポリシー情報を変更
- システムへの反映処理[8]を明に実施

という操作が要求されるため，あるプロセス実行中の一定の間だけ異なるアクセス制御ポリシーを適用するというような細粒度での動的制御は実現不可能である．

## 3 提案方式

### 3.1 基本アイデア

本稿で提案するのは，オブジェクトに対して該オブジェクトを操作可能なサブジェクトを指定するという形態を取るMACベースOSとは反対に，サブジェクトに対してあるオブジェクトを操作可能とする権限を与える，という形式を備えるシステムである．その基本的アイデアは，操作対象オブジェクトに対して“鍵”を割り当てておき，該鍵によって検証可能な権限データを提示できるサブジェクトにのみ該オブジェクトの操作を許可する，というものである．そのような権限データは一種のケーパビリティもしくはKerberosにおけるチケットに相当するものと見倣せるだろう．

このような方式であれば，鍵/権限データのオブジェクト/サブジェクトへの割り当てはシステム実行中の任意のタイミングで実施可能であるため，MACベースOSでは不可能だった細かい粒度でのアクセス権限変更を実現することが可能になる．

鍵および権限データは基本的にはOSカーネルによって作成され，オブジェクトおよびサブジェクトに付与される．権限データにはサブジェクトの素性(プロセスID等)と可能な操作種別(利用可能システムコール名および操作内容等)を含めることで，万一他の主体に該データが奪取されたとしても悪用できないように図る．また，データ全体に対して対象オブジェクトのための検証用データ[9]を付与することにより改竄を防止する．

また，4章で取り上げる例のように，オブジェクトに対する特定の操作のみの制限で十分という場合には，権限データとしてオブジェクトに付与された鍵そのものを用い，オブジェクトに対する特定操作要求にサブジェクト自身が該鍵による検証情報を付与することで同等の効果を得られるケースもあると考えている．筆者らは別論文で，プロセス間通信を保護するために送受信プロセス双方のみが保有する共通鍵を用いてデータを暗号化するという方式を提案した[3]．そのなかで鍵の共有方法は状況依存で実現する必要がある旨述べたが，本方式はシステム中での任意のプロセス対に対して安全に共通鍵を配布する汎用的な方式としての利用も視野に入れている．

### 3.2 権限データ付与可能サブジェクトの限定

当然，任意のサブジェクトが任意のオブジェクトに対する権限データを得ることができたのでは意味がないため，サブジェクトが権限データを得る際には当該サブジェクトの正当性確認のためになんら

[4] ファイル，ソケット，プロセス等，計算機システム内でなんらかの操作対象となる存在の総称
[5] もしくはドメインとも呼ぶ
[6] たとえば，システム管理者/スタッフ(システム管理者役割を実行可能なユーザ)/一般ユーザ等
[7] `/etc/security/selinux/policy/src`
[8] `make load`
[9] 共通鍵アルゴリズムの場合にはMAC，公開鍵アルゴリズムの場合にはデジタル署名

かの認証行為が必要である．その方式としては通常のログイン時認証機構を利用するのでも良いし，専用の認証機構を用意するのでも良いだろう．利用者との対話的処理により正当なユーザの下で動作するプロセスであることを証明できたサブジェクトにのみ，必要な権限データが付与されるように図ることで，ユーザとの対話的処理が不可能な各種サービス提供用デーモン類による権限データの取得は不可能となる．

認証後、別途システムで定めるポリシーに基づいて該サブジェクトによる権限データ入手の可否が判定される．そのための具体的なポリシー指定方法にはさまざまなものがあり得るが，既存 DAC ベース OS への段階的な統合の容易さを考慮すれば，たとえばユーザ/グループ ID に基づく判断を基本とし，例外的なパターンのみ明にポリシーとして指定するという方式が考えられる．

また，ログイン認証機構を利用する場合，ログイン認証後に起動されるシェル等のプロセスに対してログイン認証実施済という印を付与しておき，その情報が子プロセスにも継承されるように処理することで，以降の子プロセスによる権限データ取得時には認証行為を要求しないという実現も考えられる．このような方式であれば，通常の計算機利用形態[10]での利便性が向上することになる．このような利便性向上策を取った場合でも，最も攻撃の対象となりやすいネットワークを介したサービスを提供するデーモン類は認証処理を経ずに実行されるため，万一それらのデーモンが乗っ取られたとしても，重要な行為の実施には該計算機上でのユーザ認証が必要となるため，影響範囲は限定される．

なお，本方式ではプロセス・オブジェクト[11]に付与される鍵データが他者から読み出されないという性質が必須である．このような性質は，MAC ベースの OS では `ptrace` システムコール等による他プロセスメモリ空間へのアクセスを適切に制限することでも実現できるが，一般的な DAC ベース OS で実効性のある形での実現は困難である．そのため，権限データ付与対象となるオブジェクトがプロセスである場合には，該オブジェクト生成時等に鍵を割り当て，該鍵が他プロセスから読み出されることのないように暗号化メモリシステム [6] により該鍵保持領域を保護することを考えている．

[10]一回のログインからログアウトまでのセッション中では，ログイン・シェル/ウインドウ・マネージャの子プロセスとしてさまざまなアプリケーションを起動することで作業を実施

[11]前述のように該鍵がサブジェクトにも付与される場合にはサブジェクトも

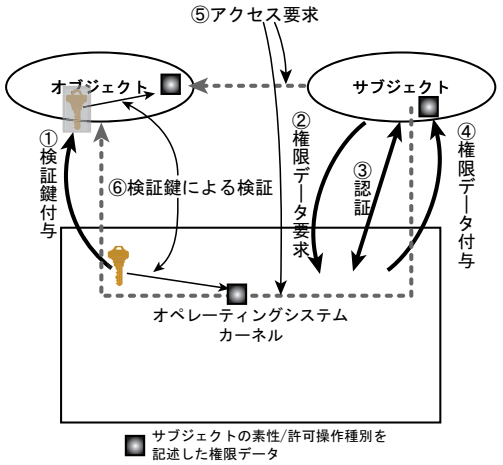


図 1: 提案方式全体像

また，プロセス以外のファイル等の静的オブジェクトへの鍵付与方法に関しては別途手当が必要となるが，それは今後の検討課題である．

### 3.3 方式全体像

本方式の全体像を図示すれば，図 1 のようになる．権限管理対象となるシステム内の重要なオブジェクト[12]に対してカーネルが鍵を付与し (図 1①)，他のプロセスが読み出すことができないような形で保存する．該オブジェクトの利用を希望するサブジェクトはカーネルに対して権限データを要求し (図 1②)，カーネルによる認証を受けることにより正規のユーザであることを証明した (図 1③) 上で権限データの付与を受ける (図 1④)．オブジェクトへのアクセス時には，サブジェクトは権限データを付与したアクセス要求をシステムコールを介してカーネル経由[13]で，もしくは，オブジェクトが処理するデータの一部として直接[14]送信する (図 1⑤)．アクセス形態に応じてカーネルもしくはオブジェクト自身が検証鍵によって該権限データを検証する (図 1⑥) ことで，該オブジェクトへのアクセスの可否が判断される．

本方式の利用形態としては以下の二種類が考えられるだろう．まず，本方式を制限強化型対策として用いる場合[15]には，本方式に基づく利用可否判定は通常のユーザ/グループ ID による許可判定の後に行

[12]システムのポリシー設定，もしくはユーザ指定によって決定

[13]ファイル・オブジェクトのように自律的でない場合

[14]プロセス・オブジェクトのように自律可能な場合

[15]通常の DAC ベース OS への導入

なう．逆に制限緩和型対策として用いる場合[16]には，MAC による許可判定の前に本方式に基づく判定を行ない，その結果が是であればアクセス可能と判定すれば良い．

特に，制限緩和型の利用をログイン認証済情報の継承と組み合わせることで，SELinux のような MAC ベース OS においても，役割を変更することなしに一時的なアクセス権限の変更が実現できるため，そのような MAC ベース OS におけるユーザビリティ向上に役立つのではないかと考えている．

# 4 試験実装

ここでは，本稿で提案した細粒度権限管理方式概念の実効性検証のために行なった試験実装の結果を述べる．本方式を完全な形で実装するには OS カーネルレベルでの対応が必要となるが，ここではまず概念の有用性の実証，すなわち，同概念が多大なオーバヘッドなしで実装可能であることを確認するために，権限管理対象を対プロセス操作に限定することでユーザランドプログラムのみによる実現とした．具体的には ssh-agent および pam_ssh と呼ばれるユーザランドプログラムに対して改造を施すことにより，たとえ root ユーザであっても他者の権限下で動作するプロセスに対する操作を実現できないという性質を備えたシステムを，CMS 実装済 FreeBSD カーネルの上に構築している．

以下ではシステム構築のために利用した ssh-agent と pam_ssh を簡単に説明した後に，改造の内容および結果を記述する．

## 4.1 利用アプリケーション

### 4.1.1 ssh-agent

ssh-agent は遠隔計算機利用システム/プロトコルである SSH[4]o におけるユーザ認証処理簡便化ツールである．ユーザは ssh-agent プロセスに自秘密鍵を預託することにより，遠隔計算機へのリモートログイン等の実施時に必要なユーザ認証処理が，該プロセスにより代行される．その際，ssh-agent は認証用データに対してユーザ秘密鍵による署名処理を行なう[17]ことで，公開鍵ベースのユーザ認証処理を実現せしめる．

一般に，ユーザ秘密鍵はパスフレーズによって保護されるため，該秘密鍵を用いた処理を実施する際に該パスフレーズの入力が求められる．ssh-agent は遠隔計算機利用時のパスフレーズ入力の手間からユーザを解放する．

### 4.1.2 pam_ssh

pam_ssh[5] は，FreeBSD や Linux，そして Solaris 等の現代的な UNIX 系 OS におけるカスタマイズ可能な認証機構 “PAM”[18]の枠組に基づき，ログイン等のユーザ認証情報としてユーザの SSH 秘密鍵パスフレーズの知／不知を利用するものである．この機能を用いると，自 SSH 秘密鍵パスフレーズを入力することによるシステム・ログイン等が可能となる．認証処理終了後，それがログイン認証のようにその後一定期間の継続的な作業を伴うものである場合，pam_ssh はさらに自動的に ssh-agent を立ち上げユーザの秘密鍵を記憶させる．

ssh-agent 単独利用では，最初に自秘密鍵を登録させるために一回だけパスフレーズを入力する必要がある．pam_ssh では，秘密鍵保護解除用のパスフレーズ入力がログイン認証と統合されるため，手元の計算機と同時に他の遠隔計算機も利用可能になるといういわゆる “Single Sign On” が実現されることになる．

### 4.1.3 改造内容

ssh-agent による認証処理代行は，当該 ssh-agent プロセスとの間で UNIX domain socket を介した通信が可能なプロセスが実施可能なため，一般的な UNIX 系 OS であれば同じユーザもしくは root 権限下で動作するプロセスが乗っ取られた場合，既存の ssh-agent プロセスを利用することで当該ユーザによって認証可能なように設定されている他の計算機にまで侵入を許すことになる．

筆者らは過去に提案した暗号化メモリシステム (CMS) で，ssh-agent のユーザ秘密鍵保持領域を保護することにより，root ユーザであっても該秘密

[16] MAC ベース OS への導入

[17] SSH protorol version 2 の場合．version 1 ではサーバから送られる暗号化済データを復号し，その結果と Session ID とを結合したものから計算した MD5 ハッシュ値を送り返す

[18] Pluggable Authentication Module

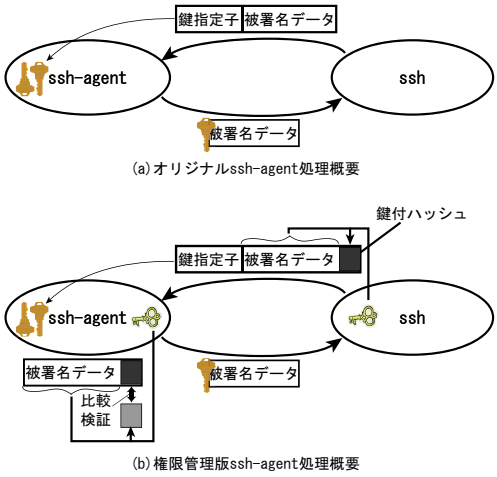


図 2: ssh-agent による署名処理概要

鍵を読み出すことが困難となるようなシステムの試験実装を行なった [6]．しかし，攻撃者にとっては目的とする秘密鍵そのものが入手できないとしても，同鍵を用いた署名等の処理を行なわせられれば十分な効果が得られるという意味で，この手当は不十分だったと言える．

本稿では，さらに ssh-agent を 3 章で提案した方式に則った形で，固有の検証鍵を保持し同鍵で検証可能なデータを送られたときにのみ署名等の重要な処理を行なうように改造することで，本来のユーザ以外による悪用は不可能となった．

関連アプリケーションに対して適用した改造は以下の通りである．

ssh-agent
: (1) プロセス起動時に疑似乱数生成器により 160bit 鍵を生成し，CMS 保護領域に保存．(2 被署名データに対する署名処理要求には，該データの後部 20byte に上記鍵を用いた鍵付ハッシュが付与されているものとして検証を行ない，正常に検証できた要求のみに署名処理を施す．(3) 他プロセスから鍵データの送付要求を受けた場合には，自らが保持する認証用鍵ペアのうち，公開鍵を用いて暗号化した上で鍵を送付する．

pam_ssh
: 通常通りに起動およびユーザ秘密鍵を記憶させた ssh-agent から暗号化された検証鍵データを読み出し秘密鍵で復号した上で，認証後に起動されるユーザシェルの環境変数として設定する．

表 1: 試験実装環境

| | |
|---|---|
| CPU | Intel Pentium III 852 MHz |
| OS | FreeBSD4.8 /w CMS サポート |
| SSH 実装 | OpenSSH 3.9p1 |
| pam_ssh 実装 | pam_ssh1.9 |

ssh
: 遠隔計算機利用アプリケーションである ssh は，認証処理を ssh-agent に依頼する際に，環境変数から得る検証鍵で生成した鍵付ハッシュを付与した被署名データを送付する．

オリジナルの ssh-agent では，UNIX domain socket を介して送られる署名依頼メッセージに対して単に署名を施して返すという処理のみが行なわれていた (図 2 (a))．しかし，本方式を実装した権限管理版 ssh-agent では，送られる署名依頼メッセージ中の被署名データの後部 20byte がそれ以外の部分に対して検証鍵による鍵付ハッシュを施した結果とみなして検証処理を行なった上で署名処理の可否が判断されるため (図 2 (b))，検証鍵を知らないプロセスは該 ssh-agent に対して署名処理をなさしめることはできない．そして，署名用鍵ペアの公開鍵を用いて暗号化された形でのみ検証鍵要求者に送信される検証鍵は，対応する秘密鍵を知る正当なユーザ以外には入手できないことが保証される．

なお，現行の CMS ではセキュリティ向上のため execve システムコールで他プログラムの実行を開始する際には保護領域の内容が継承されないようになっており，シェル側での検証鍵保持領域に対しては CMS による保護を適用していない．今回は概念の実証が目的であるためこのような実装としたが，この点に関してはなんらかの改善が必要と考えている．

## 4.2 試験実装結果

試験実装は，表 1 に示すような環境で行なった．

同試験実装により，たとえ root ユーザであっても他ユーザの秘密鍵パスフレーズを知ることができない限り同ユーザが実行中の ssh-agent を用いた公開鍵認証は行なえないことが確認できた．鍵の生成は ssh-agent プロセス起動時，また，権限データ[19]付与もログイン認証終了後に起動されるシェル・プロセスに対して動的に行なわれるという意味で，

[19] 本実装では鍵そのもの

表 2: 性能計測結果

| agent 種別 | real(sec) | user(sec) | system(sec) |
|---|---|---|---|
| オリジナル | 33.01 | 4.06 | 0.50 |
| | 33.03 | 3.95 | 0.61 |
| | 33.03 | 4.03 | 0.55 |
| | 33.03 | 4.04 | 0.51 |
| | 33.05 | 4.04 | 0.54 |
| | 33.07 | 4.02 | 0.54 |
| 権限管理版 | 33.04 | 3.97 | 0.62 |
| | 33.04 | 4.03 | 0.52 |
| | 33.05 | 3.99 | 0.61 |
| | 33.06 | 3.94 | 0.65 |
| | 33.13 | 3.95 | 0.63 |
| | 33.16 | 4.03 | 0.56 |

3 章に目的として掲げた動的変更可能な権限管理機構が `pam_ssh` の Single Sign On によるユーザビリティを損なうことなく実現されている．

また，本試験実装に基づく `ssh-agent` とオリジナルの `ssh-agent` とを用いて公開鍵認証によるリモートアクセスを行なった場合の性能差を表 2 に示す．その際に用いたクライアント側は 1 に示した通り，サーバ側は CPU: Celeron 2.40GHz，OS: Gentoo Linux (Kernel 2.4.26)，SSH デーモン: OpenSSH 3.9p1 を用いており，サーバ/クライアント間は 100base-T と switching hub で接続されている．測定はクライアントからサーバに対して `true` コマンド[20]実行要求を 100 回連続して送り，それに要する時間を計測するという形で行なわれた．表はそのような計測を各 `ssh-agent` に対して 6 回ずつ行なった結果をまとめている．

表から明らかな通り，性能の傾向に大きな差異はなく，オリジナル測定中最速値と本方式測定中最遅値とを比べてもその差は 0.5%未満であることから，本方式実装の有無は実用時における性能に対して有意な差を与えていないと判断できる．これは，本方式を適用した部位が遠隔計算機利用時におけるユーザ認証処理中の署名処理依頼というごく限定的な箇所であることからほぼ自明なことではあるが，CMS 保護によるオーバヘッドがあっても十分高速に実行できることを実証できたという点で意味があったと考える．

[20]何の処理も行なわず単に成功を返すコマンド

# 5 結論ならびに今後の課題

以上，現在の一般的な DAC ベース OS に対してより細粒度なアクセス権限管理手法を導入するための方式を提案した．また，ユーザランドにおける試験実装の結果を述べることにより，実用性能を落すことなく，かつ，ユーザビリティを損なわずに導入可能という同方式の実用性をも示すことができたと考えている．また，同方式は SELinux 等の MAC ベースの OS におけるユーザビリティ向上のために導入する可能性の指摘も行なった．

今後の課題としては，(1) カーネルレベルでの導入による本方式のさらなる実用性の検証，(2) ファイル等の静的オブジェクトへの安全な鍵，もしくは権限データ付与方式の検討，(3)MAC ベース OS の代替となる全体システムの構成検討および実装による評価，(4) 本方式で実現が期待されるポリシー設定省力化の評価，(5) 文脈依存な形でのアクセス制限実現のための動的な権限管理方式の検討等が挙げられる．

# 参考文献


[1] Bill McCarty: SELinux – NSA's Open Source Security Enhanced Linux, O'reilly, October 2004.

[2] 田端 利宏，櫻井 幸一: ファイルアクセスパーミッションの統合手法とそのトレードオフに関する考察, 2005 年 暗号と情報セキュリティシンポジウム (SCIS 2005) 予稿集, Vol. 1, pp.79-84, January 2005.

[3] 稲村 雄, 竹下 敦: 同一計算機内プロセス間通信に対する保護のための一手法, 2005 年 暗号と情報セキュリティシンポジウム (SCIS 2005) 予稿集, Vol. 1, pp.61-66, January 2005.

[4] IETF Secure Shell WG: http://www.ietf.org/html.charters/secsh-charter.html

[5] pam_ssh home page: http://sourceforge.net/projects/pam-ssh/

[6] 稲村 雄，本郷 節之: 暗号技術によるメモリデータ保護方式の提案, 情報処理学会論文誌, Vol. 45, No. 8, August 2004.