情報処理学会論文誌　コンピューティングシステム　Vol. 1　No. 2　275–284 (Aug. 2008)

SAccessor：デスクトップPCのための安全なファイルアクセス制御

滝澤裕二[†1,*1]　光来健一[†1]
千葉滋[†1]　柳澤佳里[†1,*2]

従来のシステムではOSによるアクセス制御によりファイルへの不正アクセスを防いできた．しかし，攻撃者にOSの制御を奪われた場合には任意のファイルへのアクセスを許してしまう．この問題を解決するために本稿では，デスクトップPC上で仮想計算機を用いてファイルアクセス制御をOSから分離し，実用的に使えるようにしたシステムSAccessorを提案する．SAccessorでは1台のマシン上でユーザがログインする**作業OS**とファイルサーバの役割を担う**認証OS**を動作させ，認証OSでファイルアクセス制御を行う．システムファイルの不正書き換えを防ぐために，認証OSが提供するサービスを介した一貫性のとれた書き換えを強制する．その際には，認証OSが出す認証ダイアログを用いてユーザを安全に認証する．ユーザの機密ファイルにアクセスする際には実用性を考慮して，ファイルグループ単位で認証を行い，一定期間は認証を省略できるようにした．

Secure File Access Control for Desktop PCs

Yuji Takizawa,[†1,*1] Kenichi Kourai,[†1]
Shigeru Chiba[†1] and Yoshisato Yanagisawa[†1,*2]

Traditional systems prevent illegal file accesses by the access control in operating systems (OSes). However, if OSes themselves are compromised, attackers can access arbitrary files. To solve this problem, we propose SAccessor, which is a system that practically separates file access control from an OS using virtual machines. SAccessor runs the *work OS*, which users log in, and the *authentication OS*, which provides a file server and file access control, on one machine. It enforces consistent rewrites via services provided by the authentication OS to prevent illegal rewrites of system files. At this time, the authentication OS safely authenticates users using an *authentication dialog box*. For users' confidential files, SAccessor performs authentication for a group of files and allows users to access those files without authentication for a specified period.

1. はじめに

従来のシステムではオペレーティングシステム（OS）によるファイルアクセス制御によりファイルへの不正アクセスを防いできた．しかし，OSにも脆弱性が報告されており，それらの脆弱性を利用した攻撃を受けるとOSによるアクセス制御が無効化されるおそれがある．OSへの攻撃の影響を受けずにアクセス制御を行うためには，OSの外部でアクセス制御を行う必要があるが，これは容易ではない．たとえば，ストレージデバイスでアクセス制御を行う場合，ユーザの情報やファイルシステムの情報が不足しているため，十分なアクセス制御を行うことができない．また，OSが攻撃されている可能性があるため，OSから情報を受け取ることができたとしてもそれを信用することはできない．

SAccessorはデスクトップPCにおいてファイルアクセス制御を攻撃対象のOSから分離し，上記の問題を解決して実用的に使えるようにしたシステムである．SAccessorではファイルアクセス制御を分離するために，仮想計算機（VM）を用いて1台のマシン上で作業OSと認証OSの2つのOSを動作させる．認証OSはファイルサーバの役割を担い，ファイルアクセス制御を提供する．ユーザがログインする作業OSは認証OSと通信することでのみファイルアクセスを行うことができる．このような構成をとることで，ファイル単位のアクセス制御が可能になり，作業OSがクラックされたとしても認証OSでアクセス制御を強制することができる．

システムファイルに関しては，作業OSには基本的に読み込みのみを許可し，不正な書き換えを防ぐ．システムファイルの書き換えは認証OSが提供するサービスを通してのみ可能となる．提供するサービスはsetuidプログラムと同じ粒度であるため，システムファイルの一貫性のとれた書き換えを強制することができる．サービスの起動時には認証OSが認証ダイアログを表示し，作業OSを介さずに直接ユーザ認証を行う．そのため，作業OSに侵入した攻撃者からは認証を成功させることができない．

ユーザの機密ファイルに関しては，ファイルにアクセスした際に認証ダイアログを用い

†1 東京工業大学大学院情報理工学研究科数理・計算科学専攻
Department of Mathematical and Computing Sciences, Tokyo Institute of Technology
∗1 現在，日立電子サービス株式会社
Presently with Hitachi Electronics Services Co., Ltd.
∗2 現在，日本電信電話株式会社サイバースペース研究所
Presently with NTT Cyber Space Laboratories

て認証を行う．実用性を考慮しつつ，できるだけ被害を限定できるようにするために，ファイルグループごとに認証を行う．認証に有効期間を設定することができ，有効期間内では認証は省略される．有効期間が終了したときには，仮想計算機モニタ（VMM）を介して認証 OS から作業 OS 内のファイルキャッシュを強制的にクリアすることにより，認証 OS によるファイルアクセス制御を強制する．

以下，2 章では従来のシステムについて述べ，3 章で提案するシステム SAccessor の設計について述べる．4 章では SAccessor の実装について述べ，5 章では評価のために行った実験について述べる．6 章で関連研究について述べ，7 章で本稿をまとめる．

2. 従来のファイルアクセス制御システムの問題

2.1 脅威モデル

SAccessor はデスクトップ PC（ノート PC を含む）に対する攻撃を想定している．デスクトップ PC はサーバと違い，コンピュータの専門知識のないユーザが管理していることが多い．また，ユーザのセキュリティ意識の低さや知識不足等が原因で，脆弱性に対する修正パッチが適用されていない場合もある．そのため，デスクトップ PC は攻撃の対象になりやすい．SAccessor ではデスクトップ PC においてメーラ等のアプリケーションだけでなく，OS がクラックされて制御を奪われることを想定している．OS がクラックされるとは，攻撃者に OS レベルの特権を取得されることである．

SAccessor は OS がクラックされた後の以下のようなシステムファイルの不正な書き換えを対象としている．

- システムの設定ファイル
- システムの実行ファイルや共有ライブラリ
- システムのログファイル

また，ユーザの機密ファイルへの不正アクセスも対象としている．

SAccessor では攻撃者は遠隔地におり，攻撃対象のデスクトップ PC へ物理的にアクセスすることはできないと仮定している．さらに，VMM や認証 OS と呼ばれるユーザが直接ログインしない OS は安全であると仮定する．VMM は OS よりも小さなソフトウェアであるため，攻撃を受けにくいと考えられる．また，認証 OS は PC 外との通信を制限するためリモートからの攻撃を受けにくい．

2.2 ファイルアクセス制御の分離に関する問題点

従来のファイルアクセス制御等のセキュリティ機構の大部分は OS によって実現されてきた．たとえば，UNIX は OS カーネルの持つアクセス制御リストによってユーザのファイルに対するアクセス権限を管理しており，SELinux[7] は OS カーネルの拡張モジュールとして強制アクセス制御を実装している．しかし，ローカルユーザが管理者権限を取得できる脆弱性[3] や OS カーネルをバッファオーバフローさせて任意のコードを実行させる脆弱性[4] 等，OS にも多数の脆弱性が報告されている．

このような OS の脆弱性を攻撃されると OS の制御が奪われ，OS によるファイルアクセス制御が無効化される危険性がある．UNIX では認証をバイパスして管理者権限を取得されると管理者のファイルにアクセスされてしまう．たとえ SELinux 等のセキュア OS[5),7),15)] で管理者権限を制限していても，強制アクセス制御自体が機能しなくなるので任意のファイルへのアクセスを許してしまう．

OS の脆弱性を利用した攻撃の影響を避けるためには OS の外部でファイルアクセス制御を行う必要があるが，これは容易ではない．たとえばディスク等のデバイスでアクセス制御を行うことが考えられる．また，OS を VM 内で動かし，VMM でアクセス制御を行うこともできるだろう．しかし，デバイス単体や VMM では OS の情報が不足しているため，ディスクのブロック番号レベルの抽象度の情報しか得られず，どのユーザのどのファイルに対するアクセスなのか判断することが難しい．OS からユーザの情報を受け取ってアクセス制御を行うことも考えられるが，OS がクラックされた後は OS から受け取った情報は信用することができない．

ファイルサーバを利用することでファイルアクセス制御をデスクトップ PC からリモートサーバに分離することも考えられる．この場合，ファイルシステムのマウント時に認証を行い，クライアントからのファイルアクセスのリクエストに対して，サーバ側でアクセス制御を行う．ファイルサーバへのアクセスはファイル名等，抽象度の高いインタフェースを利用することができるため，従来の OS と同等のアクセス制御を行うことができる．しかし，ファイルサーバの認証を成功させた後にクライアントに侵入された場合には任意のファイルへアクセスを許してしまう．認証前であっても，クライアントの OS をクラックされてキーロガーを仕掛けられ，パスワード等を盗まれた場合には，攻撃者がファイルサーバの認証を成功させることができる．また，ノート PC を出張先で使うような場合にはファイルサーバの利用は難しい．

3. SAccessor

我々はファイルアクセス制御を攻撃対象の OS から分離したときに生じる問題を解決し，

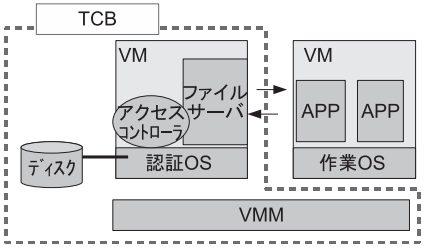


図 1　SAccessor のアーキテクチャ
Fig. 1　The architecture of SAccessor.

実用的にファイルアクセス制御を行える SAccessor を提案する．SAccessor は VM を用いてファイルアクセス制御をユーザのログインする OS から分離する．SAccessor を使えばユーザのログインする OS がクラックされた状況においても，システムファイルの不正な書き換えを防ぎ，ユーザファイルへの攻撃の被害を限定することができる．

3.1　ファイルアクセス制御の分離

SAccessor は**図 1** のように VM を用いて認証 OS，作業 OS の 2 つの OS を動作させる．作業 OS にユーザがログインし，認証 OS がファイルサーバになる．作業 OS はローカルファイルシステムを持たず，認証 OS 上に置かれた作業 OS 用のディレクトリをルートディレクトリにマウントして利用する．ファイルサーバを使うことで，作業 OS から認証 OS のディスクにファイル名等の高い抽象度でのアクセスが可能である．作業 OS はこれ以外の方法ではディスクにアクセスすることはできない．

SAccessor は作業 OS での従来のファイルアクセス制御とは別に，認証 OS でも独立してファイルアクセス制御を行う．これにより，作業 OS の制御を奪われてもファイルアクセス制御を機能させることができる．作業 OS の制御が奪われるまでは作業 OS でのアクセス制御も機能する．

3.2　システムファイルのアクセス制御

3.2.1　認証 OS によるサービス提供

システムファイルの不正な書き換えを防ぐために，作業 OS からシステムファイルへのアクセスは一部の例外を除き，読み込みのみを許可する．ここでいうシステムファイルとはインストールされたプログラム，ライブラリ，設定ファイル等である．`/var/run/xxx.pid` 等の PID を記録するファイルや，ログファイル，`/var/run/utmp` 等，システムによって自動で書き込みが行われるようなファイルについては，例外的に作業 OS による書き込みを許可する．ログファイルは追記のみを許可する．追記のみであるため，ログファイルが改ざんされることはなく，侵入の痕跡を消されることはない．pid 等は攻撃者によって書き換えられる恐れがあるが，システムを再起動した後には上書きされるため影響は残らない．

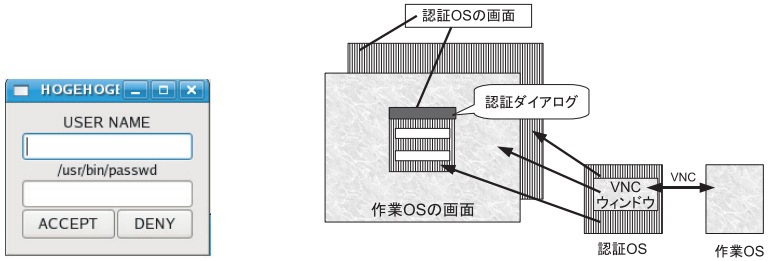
一般ユーザはシステムファイルを root に setuid されたプログラム（以下 setuid プログラム）を使って書き換えるため，認証 OS は setuid プログラム単位でサービスを提供する．つまり，認証 OS のサービスは作業 OS がシステムファイルを書き換えるためのインタフェースとなる．認証 OS 上ではサービスを制御するためのデーモンが動作しており，作業 OS からサービス起動のリクエストを受け取ると，そのサービスに対応した setuid プログラムを実行する．作業 OS からはそれらのサービスを通してのみシステムファイルの書き換えができ，直接システムファイルを書き換えることはできない．よって，たとえ作業 OS がクラックされても，作業 OS を再起動することでクラックの影響を取り除くことができる．

認証 OS の提供するサービスの例としてパスワード変更するためのサービスがある．SAccessor では実行するプログラムが setuid されていると認証 OS と通信し，ユーザを認証してから認証 OS 上の対応する setuid プログラムを実行する．このサービスの場合は，認証 OS 上の passwd プログラムを呼び出す．認証 OS 上の passwd プロセスはユーザから受け取ったデータをもとに `/etc/passwd` ファイルの書き換えを行う．このファイルは作業 OS からは読み込みのみに設定されているので，作業 OS が直接このファイルを書き換えることはできない．

3.2.2　認証ダイアログ

SAccessor では安全に認証を行うために，サービスの起動時に認証 OS が認証ダイアログを出して直接ユーザとやりとりする．**図 2** (a) は認証ダイアログのスクリーンショットである．ユーザ名とパスワードを入力させて認証を行う．認証ダイアログ中央にはリクエストされた認証 OS のサービスに対応する setuid プログラムが表示されており，ユーザの身に覚えのないサービスの実行である場合には DENY ボタンを押して認証を失敗させることができる．このダイアログは setuid プログラムの実行時のみ表示されるため，ダイアログが出る頻度は低い．

この認証ダイアログは作業 OS の画面とは独立した認証 OS のウィンドウとして，作業 OS の画面の上に重ねて表示する．認証 OS のウィンドウである認証ダイアログには作業 OS に侵入した攻撃者はアクセスすることも，ダイアログを盗み見ることもできない．ユーザと認証 OS が作業 OS を介さずに認証ダイアログを通して認証を行うことで安全に認証をすることができる．ダイアログに渡されるキー入力は，キーボードから VMM，認証 OS と渡さ



(a) 認証ダイアログ　　(b) 画面構成

図 2　認証ダイアログと画面構成
Fig. 2　An authentication dialog box and window stacking.

れる．これらは安全であると仮定しているので，このキー入力は作業 OS に侵入した攻撃者に盗まれることはなく安全に認証 OS 上のサービスに受け渡すことができる．

その際に認証ダイアログをシームレスに表示するために作業 OS 上で VNC[12] サーバを動作させ，認証 OS で VNC クライアントを動作させる．VNC とはネットワーク上の離れたコンピュータを遠隔操作するためのソフトである．まず認証 OS で VNC のウィンドウを全画面表示させ，ユーザからは VNC のウィンドウだけが見えている状態にする．次に，認証ダイアログを VNC のウィンドウの上になるように表示する（図 2 (b)）．このようにすると，ユーザには認証ダイアログがあたかも作業 OS から出されたように見える．

認証ダイアログは作業 OS の画面上に重ねて表示するため，作業 OS に侵入した攻撃者に偽のダイアログを出された場合，本物の認証ダイアログと区別するのが難しい．攻撃者によって出された偽のダイアログにパスワードを入力してしまうと，攻撃者にパスワードを盗まれてしまう．SAccessor は攻撃者が出す偽の認証ダイアログと区別するために，認証ダイアログのタイトルにはシークレット文字列を表示する．シークレット文字列はあらかじめ認証 OS 側にユーザが登録しておくものであり，作業 OS からは参照することができない．そのため，攻撃者はこのシークレット文字列を偽のダイアログに表示することが難しい．ユーザはシークレット文字列を見ることで認証ダイアログを偽のダイアログと区別することができる．

3.2.3　安全な入出力

認証 OS のサービスの入出力を安全に行えるようにするために，サービスは認証 OS のターミナルの中で実行する．このターミナルソフトのウィンドウは認証ダイアログと同様に作業 OS の VNC ウィンドウの上に表示される．setuid プログラムが GUI プログラムの場合には，そのウィンドウを作業 OS の画面の上に表示する．このウィンドウを通してユーザと認証 OS が入出力の受け渡しを行う．これは，作業 OS 経由でキー入力や画面出力を行ってしまうとユーザが起動した setuid プログラムに対して攻撃者がキー入力したり，出力を盗み見されたりする危険があるからである．たとえばパスワード変更の操作を行っているとき，攻撃者にキー入力を盗まれるとパスワードを知られてしまう．また，キー入力に割り込まれるとユーザの意図しないパスワードに変更されてしまう．

3.3　ユーザファイルのアクセス制御

システムファイルにアクセスする setuid プログラムは細心の注意を払って作成されていると考えられるため，認証 OS 側で安全に実行できる．しかし，ユーザファイルにアクセスする一般のプログラムを認証 OS で動かすのは危険である．たとえばメーラを認証 OS のサービスとして提供すると，ウィルスメールを開いた場合に悪意のあるコードが認証 OS 上で実行され，任意のファイルにアクセスされてしまうからである．

そこで，ユーザファイルに対するアクセスに対しては，作業 OS で動いているプログラムがファイルにアクセスしたときに認証ダイアログを表示して認証を行う．ダイアログの中央にはアクセスしようとしているファイル名とアクセス権が表示される．しかし，ファイルアクセスのたびに認証ダイアログによる認証を行うと，著しく利便性が落ちてしまい実用的にならない．その一方で，認証をログインのときにしか求めないようにするとログイン中に作業 OS の制御を奪われたときに，それ以降は任意のファイルにアクセスされてしまう．

3.3.1　認証の有効範囲の限定

SAccessor では，ファイルやディレクトリをまとめたグループ単位でアクセス制御を行う．ポリシファイルにグループを定義することができ，それぞれのグループに対して許可するアクセス権と認証の有効期間を設定する．1 度グループに対して認証を行ったら，認証の有効期間内ではグループのファイルへのアクセスに対する認証は省略され，認証の回数を減らすことができる．また，ログイン時のみに認証する場合と比べると，よりきめ細かいアクセス制御が可能になる．たとえ認証を行ったあとで作業 OS の制御を奪われたとしても，攻撃者は認証が有効になっているグループのファイルにしかアクセスできず，攻撃による被害を限定することができる．このように SAccessor では実用性を考慮しつつ，一定の安全性を確保することができる．

図 3 は SAccessor のポリシ例である．タグで囲まれた範囲が 1 つのグループになり，その中で羅列されるファイルのパス名がグループに属するファイルになる．それぞれのパス名

```
⟨Mailer⟩ [3600]
/home/takizawa/.thunderbird/* (rw)
⟨/Mailer⟩

⟨ssh⟩[10]
/home/takizawa/.ssh/id_rsa (r)
⟨/ssh⟩
```

図 3　SAccessor のポリシ例
Fig. 3　A policy example for SAccessor.

の後ろには許可したいアクセス権を記述する．r，w はそれぞれ読み込みと書き込みを表しており，a は追記を表す．グループタグの隣に書かれた数字は認証の有効期間（秒）を表している．有効期間を省略すると，そのグループに対しては認証を行わずアクセス制御のみを行う．

図 3 のポリシを適用した場合，ユーザがメールを見るためにメーラを起動し，認証を行うと Mailer グループに属する .thunderbird ディレクトリ以下のファイルへのアクセスが可能になる．このとき，もしウィルスを添付されたメールを受信し，ウィルスプログラムがユーザの秘密鍵 id_rsa へアクセスしようとすると，ssh グループのための認証を必要として認証ダイアログが表示される．ウィルスはこの認証を成功させることができず，ユーザがこの認証を失敗させることで id_rsa へのアクセスを防ぐことができる．

ポリシに依存するが一般的にはダイアログはそれほど出ないと考えられる．なぜならば，ユーザのファイルのほとんどはそれほど重要なファイルではなく，一部の機密ファイルのみを扱うときだけ認証ダイアログを出して認証すればよいからである．

また，ポリシの作成にはそれほど手間はかからないと考えられる．PC に保存されているファイルは関係の深いファイルがディレクトリごとに保存されているので，ポリシに定義するグループは多くの場合，ディレクトリ単位で作成することができる．認証の有効期間はグループの数だけ決めればよい．それぞれのファイルのパーミッションは作業 OS のファイルの所有者のパーミッションと同じにすればよい．

3.3.2　ファイルキャッシュの制御

作業 OS が認証 OS のファイルサーバから取得したファイルの内容はファイルキャッシュとして作業 OS 上のメモリに保持され，ファイルキャッシュが有効な間のファイルアクセスはそのファイルキャッシュに対して行われる．作業 OS にファイルキャッシュが残っていると，作業 OS は認証 OS と通信を行わずキャッシュに対してアクセスするために，認証の有効期間が終わっても認証を行わずにファイルを読み続けることができてしまう．

たとえば図 3 において，初めて id_rsa にアクセスするときには認証を求められる．認証に成功すると実際にファイルを読み込み，作業 OS のメモリにキャッシュする．認証の有効期間である 10 秒が過ぎた後に再び id_rsa を読み込もうとした場合，認証 OS と通信して認証を行うべきだが，作業 OS にファイルキャッシュが残っている場合には認証 OS とは通信せず作業 OS のファイルキャッシュに直接アクセスしてしまう．そのため認証の強制ができず，攻撃者に id_rsa ファイルを読まれる危険性が大きくなってしまう．

SAccessor ではこの問題を避けるために，認証の有効期間を過ぎたファイルのキャッシュは認証 OS が作業 OS 上のメモリ上から強制的にクリアする．認証 OS は VMM の機能を使って作業 OS のメモリからクリアすべきキャッシュを探す．認証 OS が作業 OS のメモリ操作を行うので，作業 OS の変更は不要である．作業 OS がクラックされるとうまくキャッシュをクリアできなくなるかもしれないが，それによって安全性は低下しない．クラック時に作業 OS のキャッシュ上にあるファイルは認証の有効期間内であるため，認証なしでファイルから読み込むことができるからである．

4. 実　　装

我々は SAccessor のプロトタイプの実装を行った．SAccessor のプロトタイプ実装の行数はユーザ空間のプログラムとして 3500 行程度，カーネル空間のプログラムとして 810 行程度であった．この章では SAccessor の実装に関して，アクセス制御，setuid プログラムの実行，ファイルキャッシュクリアの方法について述べる．

4.1　ファイルアクセス制御

SAccessor のファイルアクセス制御の実装は**図 4** のようになっている．VMM としては Xen [1] を使用し，認証 OS をドメイン 0 で，作業 OS をドメイン U で動作させる．両 OS には Linux を使用している．ファイルサーバには NFS を使用した．図のアクセスコントローラ，認証プロセスが SAccessor のために追加したプログラムである．また，inode 構造体に認証済みかどうかを表すフラグを追加し，認証 OS の NFSD を改造した．

NFSD は作業 OS からリクエストを受け取ったときにアクセスコントローラを呼び出す．アクセスコントローラは起動時に登録されたポリシに従ってアクセス権のチェックを行い，認証を行う必要があるときには認証プロセスと通信し，認証ダイアログを出すように命令す

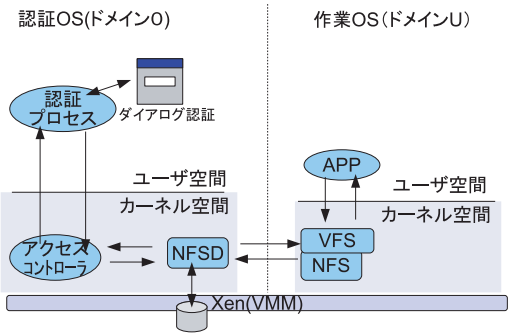


図 4　SAccessor のファイルアクセス制御
Fig. 4　The file access control in SAccessor.

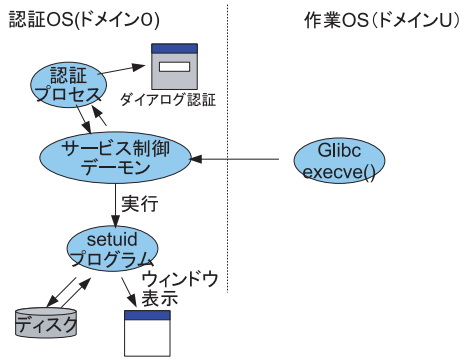


図 5　認証 OS 上での setuid プログラムの実行
Fig. 5　The execution of setuid programs on the authentication OS.

る．認証プロセスは認証ダイアログを表示してユーザ認証を行い，結果をアクセスコントローラに返す．認証ダイアログには NFS 経由で渡されたファイル名とアクセス権を表示する．認証ダイアログに表示するシークレット文字列は起動時に登録する．認証に成功した場合，あるいは認証が必要ない場合，アクセスコントローラはそのファイルに対応する `inode` 構造体に追加したフラグを真にする．このフラグが真になっているファイルに対する認証は省略される．アクセスコントローラは認証の有効期間を過ぎるとこのフラグを偽にする．

SAccessor は追記のみを許可するポリシを提供しているが，NFSD は追記かどうか判断することができない．そこで，ファイルアクセスが追記かどうかはファイルサイズと書き込みのオフセットを比較して判断する．オフセットがファイルサイズに等しければ追記とする．

4.2　setuid プログラム実行の自動転送

図 5 は setuid プログラムの実行の手順を示している．SAccessor では利便性のために作業 OS 上の glibc の `execve` 関数を改造した．作業 OS 上ではこれ以外の修正は行っていない．我々の改造した `execve` 内では setuid プログラムかどうかを `stat` 関数で識別し，自動で認証 OS 上のサービス制御デーモンと通信を行う．認証プロセスが認証ダイアログによる認証を行い，認証に成功すると，サービス制御デーモンが認証 OS 上の setuid プログラムを起動する．

glibc の改造は setuid プログラムの実行を認証 OS に自動的に転送するために行ったものである．glibc を経由せずにサービスを呼び出したとしても，認証を求められるため安全性は低下しない．

4.3　作業 OS のファイルキャッシュクリア

作業 OS のファイルキャッシュをクリアするために，認証 OS 上で Xen の機能を使って作業 OS のメモリ空間の一部を認証 OS のメモリ上にマップし，認証 OS から作業 OS のメモリを直接操作する．メモリのマッピング処理は，認証 OS からメモリを操作するたびに実施する．認証 OS から作業 OS のメモリを操作する間は作業 OS を停止させる．これは作業 OS と認証 OS でファイルキャッシュに対するアクセスの競合を避けるためである．

作業 OS 上のファイルキャッシュは，ファイルパスと作業 OS のルートディレクトリの `dentry` 構造体からたどることができる（**図 6**）．`dentry` 構造体はファイルの名前とディレクトリやファイルの相互の参照関係を管理する構造体である．まず，作業 OS の CPU レジスタの値を取得する．x86 アーキテクチャでは CPU のレジスタの値から現在実行されているプロセスに対応する `task_struct` 構造体のアドレスが分かる．`task_struct` 構造体からルートディレクトリの `dentry` 構造体を取得し，ファイルパスをもとにディレクトリを順に巡回することでキャッシュクリアの対象となるファイルの `nfs_inode` 構造体を見つけることができる．`nfs_inode` からはファイルキャッシュのアドレスを知ることができるのでキャッシュの内容をゼロクリアする．また，`nfs_inode` にはキャッシュが有効かを示すフラグが存在するので，そのフラグの値を書き換えてキャッシュを無効化する．これにより NFS はゼロクリアされたキャッシュを使わず，サーバからファイルを読み直す．

4.4　SAccessor の OS 依存性

ファイルキャッシュの実装は，OS によってメモリの配置や使用している構造体が異なる

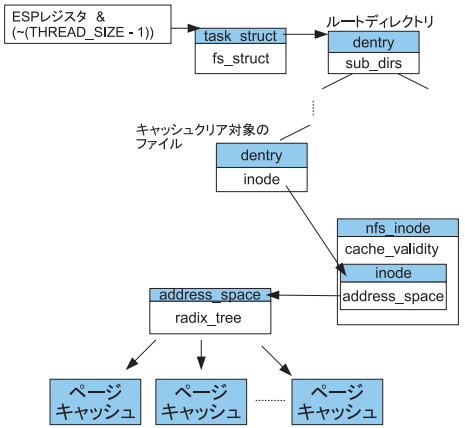


図 6　ファイルキャッシュのクリア
Fig. 6　Clearing the file cache.

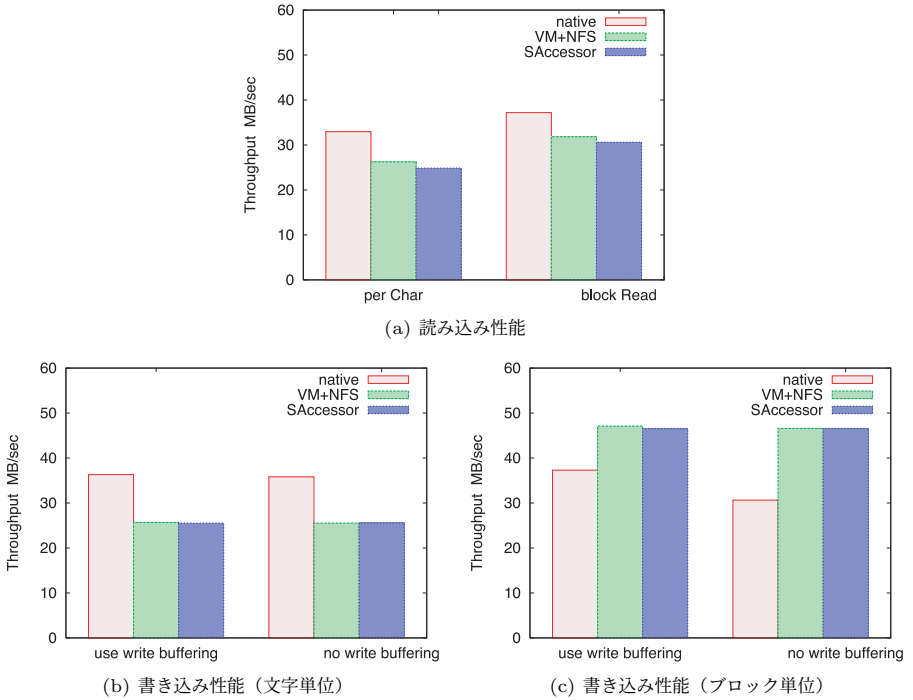
ため，OS ごとに個別の実装を行う必要がある．しかし，認証 OS から作業 OS のキャッシュを操作することはどの OS でも可能だと考えられる．認証 OS が提供するサービスは作業 OS に依存する．現在の実装のように作業 OS と認証 OS が同じであれば，作業 OS で実行する setuid プログラムと同じプログラムを認証 OS 上で実行可能である．認証 OS と作業 OS が異なる場合には作業 OS 上の setuid プログラムをエミュレートする必要がある．

5. 実　　験

我々は SAccessor を使用したときにどの程度のオーバヘッドが生じるかを測定する実験を行った．実験対象の計算機として，PentiumD 3.0 GHz の CPU，メモリ 1 GB の計算機を使用した．VMM としては Xen 3.0.4，その上で動く OS には Linux 2.6.16.33 を用いた．認証 OS を動かす VM にはメモリを 512 MB，作業 OS を動かす VM にはメモリを 384 MB 割り当てた．

5.1　ファイルアクセスの性能

ベンチマークソフト bonnie++ [2] を用いて読み込みと書き込みのスループットの測定を行った．bonnie++の使うファイルにはアクセス制限はかからないようにした 1000 行のポリシを使用した．この実験は (1) VM を用いずにローカルで ext3 ファイルシステムを使用した場合（native），(2) VM 間で NFS を用いた場合（VM+NFS），(3) SAccessor を使用した場合について行った．ファイルサイズとしては 2 G バイトを指定し，write バッファを使用する場合と，`-b` オプションを指定して write バッファを使用しない場合を測定した．文字単位の I/O では `putc`，`getc` 関数を使い，ブロック単位の場合は `write`，`read` システムコールを用いて測定を行っている．



(a) 読み込み性能
(b) 書き込み性能（文字単位）
(c) 書き込み性能（ブロック単位）

図 7　ファイルアクセス性能
Fig. 7　The performance of file accesses.

実験結果は**図 7** のようになった．読み込みのスループットが最悪になるのは文字単位で読み込みを行った場合であった．SAccessor と native を比較した場合，約 33%のオーバヘッドとなった．native と VM+NFS を比較した場合は約 26%のオーバヘッドになることから，SAccessor のオーバヘッドのうち 26%が VMM と NFS を利用していることで，残りの 7%が SAccessor で新たにアクセス制御を行うために追加した処理の時間であると考えられる．

書き込みのスループットが最悪になる場合は文字単位で書き込みを行った場合であった．SAc-

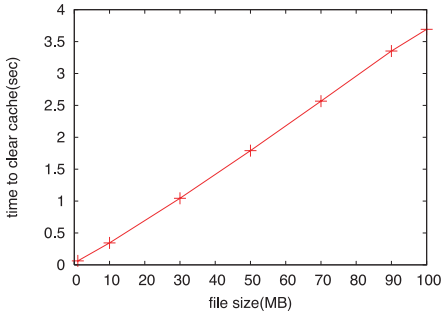


図 8　キャッシュクリア性能
Fig. 8　The performance of cache clear.

cessor と native を比較したときで約 42%のオーバヘッドになった．SAccessor は VM+NFS と比べて約 0.4%のオーバヘッドがあった．一方，ブロック単位の書き込み性能では，native よりも SAccessor の方が性能が良くなった．これはローカルファイルシステムでは，ディスクへ書き込みが終了した時点で実験が終了となるが，NFS を使った場合にはディスクにデータを書き込み終わった時点ではなく，NFS サーバにすべてのデータを送った時点で実験が終了するためと考えられる．

5.2　ファイルキャッシュクリアの性能

ファイルキャッシュのクリアにかかる時間を調べるために，作業 OS 上でファイルを 1 度読み込んだ後，認証 OS 側からファイルキャッシュのクリアを行った．キャッシュクリアのために作業 OS を停止させてから，キャッシュをクリアして作業 OS を再開させるまでの時間を測定した．クリアするファイルの数は 1 つである．実験結果は**図 8** のようになった．ファイルサイズに比例して必要な時間が増え，10 MB では約 0.4 秒，100 MB では約 3.7 秒かかった．ただし，キャッシュクリア時に作業 OS は停止するが，その間のマウスイベントやキーストロークの取りこぼしはなかった．

このオーバヘッドの内訳を調べると作業 OS のメモリを認証 OS に 1 ページ分マップするのに約 0.04 ミリ秒かかっていることが分かった．キャッシュを 1 ページ探すのに 3 回メモリマップを行う必要がある．ページサイズは 4 KB であるため 100 MB では約 75,000 回のメモリマップを行う必要があり，これにかかる時間は約 3 秒である．SAccessor のキャッシュクリアの実装を VMM の中に実装することでメモリマップのオーバヘッドを削減でき，キャッシュクリアの性能を大幅に改善できると考えられる．

表 1　setuid プログラムの分類
Table 1　The types of setuid programs.

分類	コマンド
作業 OS で実行しないと意味がない	mount, unmount, Xorg, newgrp, kpac_dhcp_helper, newvc, usernetctl, kgrantpty, userisdnctl
ネットワークを使うので危険	ping, ping6, rcp, rlogin, rsh
任意のプログラムを実行できて危険	su, sudo, sudoedit, kon, newrole, suexec, ksu
認証 OS 上で実行できる	change, at, chfn, chsh, gpasswd, crontab, passwd, lppasswd, pam_timestamp_check, unix_chkpwd, userhelper

5.3　サービス呼び出しの頻度

我々は認証 OS のサービスとして実行できる setuid プログラムがどの程度あるか調査した．調査対象は Fedora Core 5 である．結果は**表 1** のようになった．setuid プログラムを (1) 作業 OS 上で実行しなければ意味がないもの，(2) ネットワークにアクセスして危険なために認証 OS 上で実行するべきでないもの，(3) 任意のプログラムを実行できて危険なため認証 OS 上で実行するべきでないもの，(4) その他の認証 OS 上で実行できるものに分類することができた．

(1)，(2)，(3) は作業 OS 上で実行するため，認証ダイアログは出ない．(4) は認証 OS の提供するサービスになり，利用時には認証ダイアログが出る．認証 OS 上で実行すべき setuid プログラムでよく使われそうなものはパスワード変更やタスクスケジューラであり，使用頻度は少ないため認証ダイアログで認証する頻度は少なくて済むと考えられる．

5.4　描 画 性 能

VNC を使うことにより，描画性能にどの程度影響が出るか調べるために x11perf を用いて性能を測定した．500 × 500 の塗りつぶされた正方形を描画する場合（rect500），ピックスマップからウィンドウへ 500 × 500 の正方形をコピーする場合（copy_pixmap），500 × 500 の正方形のイメージを画面に転送する場合（putimage）を測定した．比較対象は X11org 7.0（native）と SAccessor において VNC を用いて作業 OS の画面を描画した場合について実験を行った．

結果は copy_pixmap で最悪になり，native と比べて SAccessor では約 295%のオーバヘッドがあった．rect500 の場合では，native と比べて SAccessor では約 172%のオーバヘッドになった．putimage の場合には逆に SAccessor が native に比べ約 3 倍の性能になった．この結果より，SAccessor では動画等の高度な描画性能を必要とするソフトウェアを使用することは困難であるが，VT-d [6] や IOV [9),10] 等を利用したビデオデバイスの仮想化技術が進

歩すれば VNC は不要になると考えている．

6. 関連研究

Plan9[11] はファイルサーバを用いているが，管理者権限を必要とするファイルにアクセスするためには，ファイルサーバの物理的なコンソールから操作する必要がある．これは管理者権限を必要とするファイルに対してアクセス制御と認証を分離していることになるが，管理者権限を必要とするたびにファイルサーバの前まで行って操作するのは不便である．SAccessor ではユーザの利便性を向上させるために，デスクトップ PC 上で認証ダイアログを用いたシームレスな認証を行い，setuid プログラムを安全に実行できるようにしている．

SVFS[16] もユーザの作業とファイルアクセス制御を別々の VM で行うシステムである．SVFS も 1 台のマシン上に Normal VM と Admin VM，DVM の 3 つの VM を動作させる．ユーザは通常 Normal VM にログインし，DVM がファイルサーバになる．SVFS では VM ごとに異なるアクセス権限を与える．Normal VM から重要ファイルへのアクセスは読み込みのみに制限され，重要ファイルへの書き込み権限は Admin VM に限られる．そのため，重要ファイルの書き換えをともなう管理タスクをユーザが行う場合は，通常作業を行う VM とは別の Admin VM にログインして行わなければならない．物理的には同一マシン上なので Plan9 よりは利便性が高いが，十分とはいえない．

Proxos[14] は通常のプロセスを実行する Commodity VM とは別に Private VM と呼ばれる VM を用意し，重要なファイルにアクセスするプロセスを Private VM 上に隔離して実行する．Private VM 上で実行されるプロセスによる重要なファイルへのアクセスは Private VM で実行し，それ以外の操作は Commodity VM 上の対応するシャドウプロセスに転送する．入出力は SAccessor と同様に Private VM に直接受け渡す．この方法により，重要なファイルにアクセスするプロセスが悪意のあるプロセスにクラックされることを防ぎ，重要なファイルを守ることができる．しかし，すべてのプログラムに対してどのシステムコールを Private VM で実行するのかというポリシをプログラマが記述しなければならない．

Kerberos[8),13)] 認証は鍵配布サーバと呼ばれる信頼できるサーバを用意し，そのサーバでユーザ認証を行うことで期限つきのチケット認可チケットを受け取る．ユーザはこのチケット認可チケットを使い，サービスを受けるためのチケットを受け取る．このチケットには秘密鍵が含まれ，有効期間はチケット認可チケットに従う．チケットに有効期間を設定することでチケットを盗まれたときの被害を限定するところは SAccessor と似ているが，Kerberos ではクライアントにキーロガーを仕掛けられてパスワードが漏洩したり，秘密鍵を盗まれてしまったりすると攻撃者が認証を成功させることができる．SAccessor ではクライアントの OS がクラックされたとしても，攻撃者が認証ダイアログにアクセスできないために，なりすまして認証される危険はない．

RPC や SSH 等を使えばアクセス制御やプロセスの実行をサーバ側で行うことができるが，認証やキー入力がユーザの直接扱う OS を経由するため，攻撃者に OS をクラックされてしまうと，攻撃者に任意の操作を許してしまう．SAccessor では認証とキー入力は攻撃対象の OS を介さずに行えるため，攻撃者に OS をクラックされたとしても被害は限定される．

7. まとめ

本稿では VM を用いてファイルアクセス制御を攻撃対象の OS から分離し，実用的に使えるようにしたシステム SAccessor を提案した．SAccessor では 1 台のマシン上で作業 OS と認証 OS を動作させ，ファイルサーバの役割を担う認証 OS でファイルアクセス制御を行う．システムファイルの書き換えは認証 OS が提供するサービスを通してのみ許可し，サービスを使う際には認証ダイアログを用いて作業 OS を介さずにユーザを直接認証する．また，ユーザの機密ファイルにアクセスする際には認証の有効範囲を設定することで認証の頻度を抑えつつ，被害を限定できるようにした．今後の課題は作業 OS 上のファイルキャッシュをゼロクリアする実装を VMM の中で行い，オーバヘッドを削減することである．

謝辞　研究に関して適切な助言をくださった千葉研究室の方々および本稿の執筆にあたり有益な助言をくださった査読者の方々に感謝いたします．

参考文献

1) Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the Art of Virtualization, *Proc. Symp. Operating Systems Principles* (*SOSP'03*), pp.164–177 (2003).
2) Coker, R.: bonnie++. http://www.coker.com.au/bonnie++/
3) CVE: CVE-2005-1263.
4) CVE: CVE-2005-1750.
5) Fischer-Hübner, S. and Ott, A.: From a Formal Privacy Model to its Implementation, *Proc. National Information Systems Security Conf.* (*NISSC'98*) (1998).
6) Intel Corp.: Intel Virtualization Technology for Directed I/O Architecture Specification (2006).
7) Loscocco, P. and Smalley, S.: Integrating Flexible Support for Security Policies into the Linux Operating System, *Proc. FREENIX Track: 2001 USENIX Annual*

Technical Conf. (*FREENIX'01*), pp.29–42 (2001).
8) Miller, S., Neuman, B., Schiller, J. and Saltzer, J.: Kerberos Authentication and Authorization System, Project Athena Technical Plan, Section E.2.1 (1988).
9) PCI-SIG: Address Translation Services 1.0 Specification (2007).
10) PCI-SIG: Single Root I/O Virtualization 1.0 Specification (2007).
11) Pike, R., Presotto, D., Thompson, K. and Trickey, H.: Plan 9 from Bell Labs, *Proc. Summer 1990 UKUUG Conf.*, pp.1–9 (1990).
12) RealVNC Ltd.: RealVNC Remote Control Software. http://www.realvnc.com/
13) Steiner, J., Neuman, B. and Schiller, J.: Kerberos: An Authentication Service for Open Network Systems, *Proc. Winter 1988 USENIX Conf.*, pp.191–202 (1988).
14) Ta-Min, R., Litty, L. and Lie, D.: Splitting Interfaces: Making Trust between Applications and Operating Systems Configurable, *Proc. Symp. Operating Systems Design and Implementation* (*OSDI'06*), pp.279–292 (2006).
15) Xie, H. and Biondi, P.: LIDS. http://www.lids.org/
16) Zhao, X., Borders, K. and Prakash, A.: Towards Protecting Sensitive Files in a Compromised System, *Proc. Int'l Security in Storage Workshop* (*SISW'05*), pp.21–28 (2005).

(平成 20 年 1 月 29 日受付)
(平成 20 年 4 月 29 日採録)

滝澤　裕二

2006 年東京工業大学理学部情報科学科卒業．2008 年同大学大学院情報理工学研究科数理・計算科学専攻修士課程修了．現在，日立電子サービス株式会社勤務．

光来　健一（正会員）

2002 年東京大学大学院理学系研究科情報科学専攻博士課程修了．同年日本電信電話株式会社入社．現在，東京工業大学大学院情報理工学研究科数理・計算科学専攻助教．博士（理学）．オペレーティングシステムの研究に従事．

千葉　　滋（正会員）

1991 年東京大学理学部情報科学科卒業．1996 年同大学大学院理学系研究科情報科学専攻博士課程退学．東京大学助手，筑波大学講師を経て，現在，東京工業大学大学院情報理工学研究科教授．博士（理学）．システムソフトウェアの研究に従事．

柳澤　佳里（正会員）

2003 年東京工業大学理学部情報科学科卒業．2005 年同大学大学院情報理工学研究科数理・計算科学専攻修士課程修了．2008 年同博士課程修了．現在，日本電信電話株式会社勤務．博士（理学）．オペレーティングシステム，言語処理系等の研究に従事．