情報処理学会論文誌　コンピューティングシステム　Vol. 2　No. 3　173–188 (Sep. 2009)

アプリケーションデータを保護するための VMMに基づくアーキテクチャ

尾上浩一[†1]　大山恵弘[†2]　米澤明憲[†1]

サンドボックスシステムやアンチウィルスツールのような OS やアプリケーションレベルで稼働するセキュリティシステムは，信頼できないプログラムの振舞いを制御したりメモリ上やファイル内のデータ漏洩や改竄を防止したりすることができる．しかし，攻撃者は同一実行空間で稼働する OS カーネルや特権レベルで稼働するプログラムを奪取できれば，セキュリティシステムが保護するプログラムやセキュリティシステム自体を攻撃することは困難ではない．同一実行空間で稼働する奪取されたプログラムからの攻撃を防ぐために，我々は VMM の特長である OS カーネルより高い特権レベルで動作する VMM による物理計算資源の操作と VM 単位の隔離を応用する．本論文では，VM の外側から保護したいプログラム（保護対象プログラム）に関連するデータを保護するアーキテクチャ*Shadowall* を提案する．Shadowall は OS カーネルを含む信頼できないプログラムによる，保護対象プログラムのデータの漏洩や改竄を防ぐことができる．メモリ上のデータを保護するために，Shadowall は動作レベル（ユーザレベル・カーネルレベル）の違いに基づき，VMM 層で物理メモリ上のプログラムデータを多重化する．この VMM 層でのメモリ管理操作は VM 内の OS カーネルに意識させることなく行われる．仮想ディスク上のデータを保護するために，我々は実行ファイルや設定ファイル等の保護対象プログラムに関連するファイルを保護対象プログラムが稼働する VM とは異なる VM（制御 VM）で管理する．保護対象プログラムが保護対象のファイルに関するシステムコールを発行したときに，VMM と制御 VM によりそのファイル操作をエミュレートする．我々は AMD64 アーキテクチャ上で Xen を用いて Shadowall を設計・実装し，その評価を行った．

VMM-based Architecture for Protecting Application Data

Koichi Onoue,[†1] Yoshihiro Oyama[†2]
and Akinori Yonezawa[†1]

Security systems running at the OS-level and the application-level, such as sandboxing systems and anti-virus tools, can monitor and control behaviors of untrusted programs and prevent attackers from tampering with computing resources. However, if the attackers can compromise the OS kernels or the privileged programs, it is not difficult for the attackers to compromise other untrusted programs in the same execution space and tamper with data related to untrusted programs. To prevent attacks from the compromised OS kernels and the compromised privileged programs, we use a virtual machine monitor (VMM) that can control physical resources at the higher privileged level than OS kernels (guest OS kernels) and provides strong isolation among virtual machines (VMs). In this paper, we propose *Shadowall*, a VMM-based architecture that protects data related to target programs from outside of untrusted VMs. Shadowall can prevent the compromised OS kernels and privileged programs from leaking and tampering with the target program data on memory and a virtual disk. To protect target program data allocated on the guest physical memory, the VMM provides different views of physical memory which depend on whether the target programs are running at the user level or at the kernel level. The VMM achieves this without making the guest OS aware of the translation. To protect files related to the target programs on a virtual disk such as executables and configuration files (target files), the VMM and monitor programs running in the other VM controls the target files. When the target programs manipulate the contents of the target files, the VMM and the monitor programs emulate the operations. These mechanisms to protect the target program data on memory and a virtual disk can prevent malicious programs running in the same execution space from leaking and tampering with the target program data. We implemented Shadowall by using Xen on an AMD64 architecture and evaluated it.

1. はじめに

これまでプロセスの振舞い制御[25),29)]，メモリ上のデータ追跡による攻撃検出[27),31)]やファイルの改竄検出[18),23)]等 OS やアプリケーションレベルでプログラムを保護するセキュリティシステムが提案されている．これらのセキュリティシステムによる安全性は，OS カーネルや管理者権限で稼働するプログラムが信頼できるという仮定のうえでもたらされている．このため，悪意ある者が OS カーネルや管理者権限で稼働するプログラムを奪取できた場合，脆弱性の有無にかかわらず，任意のプログラムに対して不正操作を行うことができる．たとえば，メモリ・ディスク上のデータの不正な読み出しや改竄を行うことができる．

†1 東京大学大学院情報理工学系研究科コンピュータ科学専攻
Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo
†2 電気通信大学電気通信学部情報工学科
Department of Computer Science, Faculty of Electro-Communications, The University of Electro-Communications

© 2009 Information Processing Society of Japan

メモリ・ディスク上の任意のデータを不正操作できるのであれば，悪意ある者は，セキュリティシステムの利用の有無にかかわらず，プログラムを奪取したり，プログラムが利用する機密データを不正操作したりできる．さらに，メモリ・ディスク上のデータの不正操作は，利用者やセキュリティシステムに攻撃を気づかせないためにも悪用されうる．

OS カーネルが信頼できない状況でユーザプログラムの保護を実現する方法として，仮想マシンモニタ（VMM）に基づく方法[13),28)] やマイクロカーネルに基づく方法[15)] が提案されている．これらの方法では，VMM やマイクロカーネルが提供する VM 単位の隔離を利用し，保護したいプログラム（保護対象プログラム）を，信頼できない VM とは異なる VM で稼働させる．これにより，信頼できない VM 内で稼働するプログラムが保護対象プログラムを直接攻撃することを困難にする．しかし，この方法では，1 つのプログラムの隔離のために VM 単位の計算資源を消費したり[13)]，保護対象プログラムのソースコードを修正したりしなければならない[15),28)] 等の問題が生じる．また，複数のハードウェアを利用し，ハードウェアレベルの隔離によるアプリケーションの保護も考えられる．しかし，これはハードウェアの導入・管理コストがかかる．

本論文では，保護対象プログラムを信頼できない OS カーネルを含む他のプログラムと同一実行空間で稼働させながらも，保護対象プログラムのメモリ・ディスク上のデータを保護することが可能なアーキテクチャ Shadowall を提案する．Shadowall は，保護対象プログラムに関連するメモリ・ディスク上のデータへの操作に，保護対象プログラムが稼働している VM（保護対象 VM）の外側から介入し，それらのデータの漏洩や改竄を防止する．我々は，まず保護対象プログラムのメモリ上のデータを保護するために，OS カーネルの下位層の VMM による物理資源操作を応用する．さらに保護対象プログラムに関連する仮想ディスク上のデータを保護するため，保護対象 VM 内で稼働する OS カーネルが迂回できない特権命令の捕捉と VM 単位の隔離を応用する．Shadowall では，どのプログラムを保護対象プログラムとするかやどのようにメモリ・仮想ディスク上のデータを保護するかについては，利用者がセキュリティポリシで指示する．

メモリ上のデータを保護するために，Shadowall では VMM 層の処理により，保護対象 OS カーネルに意識させることなく，ユーザレベルとカーネルレベルの動作レベルごとに保護対象プログラムが操作するメモリ上のデータが異なるようにする．動作レベルに基づいて異なるメモリ上のデータを操作させるために，暗号化技術を用いている既存のシステム[11),20),32)] とは異なり，Shadowall は 1 つの仮想アドレスに対して異なる物理ページを割り当てる．これにより，たとえ OS カーネルを含め他のプログラムが保護対象プログラムに関するメモリ上のデータに対し不正操作を行ったとしても，保護対象プログラムのユーザレベルの動作時のメモリ上のデータに影響はなく，データの漏洩および改竄を防止することができる．さらに仮想ディスク上のデータを保護するために，Shadowall は，保護対象 VM とは異なる VM（制御 VM）で，実行ファイル等の保護対象プログラムに関連するファイル（保護対象ファイル）を管理する．保護対象ファイルの中身に対する操作は，VMM 層でのシステムコール捕捉に基づいて，VMM と制御 VM によってエミュレートされる．保護対象ファイルのデータは VMM および制御 VM によって管理・操作されるため，保護対象ファイルのデータが漏洩したり改竄されたりすることを防止できる．加えて，保護対象ファイルに対する相対パスやシンボリックリンクを利用した攻撃も防ぐことができる．

我々は保護対象プログラムとして既存のウェブサーバやアンチウィルスツール等のアプリケーションを想定している．さらに，Shadowall は政府や企業等で独自に開発され，外部に公開せず運用するプログラムに対しても有用であると考えている．独自に開発されるプログラムは，すでにプログラムコードが公開されている既存のアプリケーションとは異なり，攻撃目的でのプログラム解析を防ぐためにデータの機密性を保つことも重要となる．機密性を保ちたいプログラムを Shadowall によって保護することで，攻撃者がプログラムのメモリ・ディスク上のデータを解析できなくなる．

我々は，VMM として Xen[8)] を，保護対象 OS として Linux を用いて Shadowall の設計および実装を行った．Shadowall の評価では，保護対象プログラムのメモリ・ディスク上のデータに関する漏洩と改竄が防げることの確認と，Shadowall により生じる実行時のオーバヘッドを計測した．

以降，本論文は 2 章で保護対象プログラムに関連するデータの保護手法について述べた後，3 章で Shadowall に関して説明する．4 章で Shadowall の実装について述べる．さらに，5 章で Shadowall を評価し，関連研究を 6 章で述べる．7 章でまとめと今後の課題について述べる．

2. アプリケーションに関連したデータを保護する手法

OS レベルで，保護対象プログラムに関連したデータを保護するために利用可能なセキュリティシステムがこれまで数多く提案されている[14),16),18),23),25),27),29),31)]．また，OpenSSH や Apache 等の既存のプログラムや Privtrans[9)] では，管理者権限が奪取される可能性を小さくするため，privilege separation の仕組みが提案されている．これらの手法は OS カーネルを含む同一実行空間内のプログラムが信頼できることを前提としている．しかし，OS カー

© 2009 Information Processing Society of Japan

ネル[4),5)] や管理者権限を有するプログラム[1),7)]，ライブラリ[2),6)] も脆弱性を持ちうる．このため，攻撃者がこれらの脆弱性を利用しシステムを奪取できた場合には，たとえセキュリティシステムが用いられていたとしても，攻撃者は保護対象プログラムの奪取やそれらに関連した機密情報の漏洩等の不正操作を行うことは困難ではない．また，Janus[14)] や Systrace[25)] のように保護対象プログラムの振舞いのみを制御する場合には，それらの制御対象ではないプログラムの操作は制御できない．たとえば，攻撃者が制御対象ではないプログラムを奪取し，Janus や Systrace で用いられる保護対象プログラムに対するセキュリティポリシを無効化し，メモリ・ディスク上のデータを改竄することができる．

同一実行空間で稼働する信頼できない OS カーネルを含むプログラムから保護対象プログラムに関連したデータを保護する 1 つの手法は，VMM を利用し，異なる VM（制御 VM）で保護対象プログラムを稼働させることである．たとえば，Terra[13)] では，保護対象プログラムの Trusted Computing Base（TCB）を削減するため，保護対象 VM とは異なる VM で保護対象プログラムを稼働させる．しかし，この VM 単位の隔離を利用した方法では，保護対象プログラムを稼働させるためだけに VM 単位の計算資源が消費されてしまう．保護対象プログラムが複数存在する場合には，異なる保護対象プログラム間の安全性を保護対象プログラム・保護対象外のプログラム間の安全性と同程度に保つためには，保護対象プログラムごとに VM を割り当てることが必要となり，さらに計算資源を消費することになる．さらに，機能が制限された制御 VM 内で保護対象プログラムを稼働させるため，保護対象プログラムで可能な操作が制限されてしまう．また，Nizza[15)] や Proxos[28)] では，保護対象プログラムの TCB を削減するために，security-sensitive データを操作する保護対象プログラムのコンポーネントのみ，異なる VM で稼働させる．しかし，これらのシステムを用いる場合には保護対象プログラムのソースコードを修正する必要がある．

本論文では，信頼できない OS カーネルを含む同一実行空間で稼働するプログラムから保護対象プログラムに関連するデータを保護し，かつ VM 単位の隔離を用いた場合生じる問題を改善することを目標とする．我々は，VMM によるメモリ管理処理の拡張とシステムコール捕捉，異なる VM での保護対象プログラムに関連するファイル操作の制御によってこの目標を実現する，これにより，既存の保護対象プログラムが信頼できないプログラムと同一実行空間で稼働するにもかかわらず，保護対象プログラムに関連するデータの漏洩や改竄を防止しながら保護対象プログラムを運用することができる．

3. 提案アーキテクチャ：Shadowall

3.1 脅威モデル

Shadowall における脅威モデルは，保護対象プログラムのユーザ空間のメモリ領域と保護対象プログラムに関連するディスク上のデータを直接操作し，それらのデータを漏洩させたり改竄したりする攻撃である．攻撃者は OS カーネルや管理者権限を必要とするプログラム，または保護対象プログラムに対する操作権限を有するプログラムを奪取できれば，保護対象プログラムの脆弱性の有無に依存することなく，保護対象プログラムに関連するメモリ・ディスク上のデータを不正に操作できる．メモリ・ディスク上のデータを不正に操作できれば，攻撃者は保護対象プログラムの制御の奪取やそれらが利用する機密データの漏洩・改竄といった攻撃を実現できる．さらに，メモリ・ディスク上のデータの不正操作は，利用者やセキュリティシステムから攻撃を隠蔽するするためにも悪用される．攻撃を気づかれてしまっては，利用者に奪取したプログラムの利用の停止等の対応をとられてしまうため，攻撃者にとっては不正操作を隠蔽することも重要となる．

保護対象プログラムに関するデータの漏洩や改竄，攻撃者の不正操作の隠蔽に関する攻撃の例を以下にいくつかあげる．

- 保護対象プログラムの設定ファイル，ポリシファイルやデータベースファイルの中身を改竄することにより，本来機密であるはずのデータを漏洩させる攻撃．
- メモリ上のリターンアドレスや関数ポインタ等の control data や実効ユーザ ID や認証済かどうかを表す変数等の non-control data[10)] を改竄することにより，プログラムの制御を奪取する攻撃．
- mimicry attack[22),30)] のように，セキュリティポリシを巧妙に迂回するために実行時のメモリ上のデータを改竄する攻撃．

攻撃者はこれらの攻撃を必要時に一時的に行うことで，攻撃の検出を困難にさせることもできる．攻撃者が任意のプログラムのメモリ上のデータを漏洩させたり改竄したりすることは，デバッガ等で用いられるプロセス制御 API（Linux における ptrace 等），カーネルモジュール，または特殊デバイスファイル（/dev/mem）等を利用することで容易にできる．

一方，以下のような攻撃は本論文では対象としていない．

- バッファオーバフロー脆弱性等の保護対象プログラム自体の脆弱性を利用してプロセスの振舞いを不正に変更する攻撃．
- カーネル空間で保護対象プログラムに関連する振舞いやデータ不正操作する攻撃．たと

© 2009 Information Processing Society of Japan

えば，カーネルレベルでのネットワーク通信の奪取や，保護対象プログラムをスケジューリング対象から外したり，システムコールに応答しなかったりするような DoS 攻撃.

- カーネルレベルルートキットを用いて，攻撃者がカーネルレベルの処理によって自身を隠蔽する攻撃.

保護対象プログラム自体の脆弱性を利用した攻撃に対しては，我々はサンドボックスシステムによってプログラムの振舞いを制限することでこの攻撃を緩和できると考えている．実際，我々は VM の外側からプロセスが発行するシステムコールを制御する仕組み[35] と Shadowall を組み合わせたセキュリティシステムも構築している．

3.2　Shadowall の構成

提案するアーキテクチャShadowall を**図 1** に示す．Shadowall は VMM 内コンポーネント *SW-core* と制御 VM 内の制御プログラム群からなる．Shadowall における TCB は VMM と制御 VM である．保護対象プログラムのメモリ上のデータ保護は SW-core が実現する．一方，ディスク上のデータ保護は SW-core と制御 VM 内の制御プログラムが連携して実現する．保護対象ファイルは制御 VM で実体（real ファイル）を管理し，real ファイルに対応したダミー（dummy ファイル）を保護対象 VM 内に配置する．セキュリティポリシは制御 VM で管理され，保護対象 VM の利用者，または利用者に保護対象プログラムの制御を

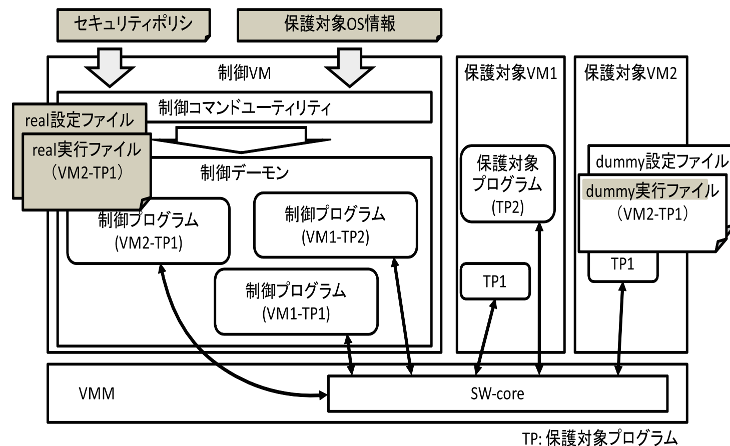

図 1　提案するアーキテクチャ：Shadowall
Fig. 1　Proposed architecture: Shadowall.

委譲された制御 VM の管理者が記述する．

Shadowall は保護対象 VM 内の OS カーネル（保護対象 OS）におけるプロセスやシステムコール等に関する情報（保護対象 OS 情報）を用いて，プログラム単位でメモリ・仮想ディスク上のデータを保護する．保護対象 VM の外側から VM 内のプロセスやシステムコールを制御する方法に関しては，文献 35) で述べているため本論文では省略する．

保護対象プログラムに対するユーザ ID やグループ ID によるロールベースのアクセス制御（RBAC）は本論文では対象としていない．Shadowall は保護対象プログラムが関連するメモリ・仮想ディスク上のデータを保護するシステムであり，RBAC とは補完関係にある．我々は，文献 35) の仕組みを併用することで，システムコール単位で保護対象プログラムのユーザ ID やグループ ID に関する制御に対応することを想定している．

3.3　Shadowall の運用

Shadowall では，保護対象プログラムのメモリ・ファイルに関する保護の設定は制御 VM から実行する．このため，攻撃者が保護対象 OS カーネルを奪取したとしても，どのプログラムを保護対象プログラムとするか，保護対象のメモリ・仮想ディスクをどのように保護するか等の保護対象プログラムに関する設定は保護対象 VM 内から行えない．また，Shadowall に適用するために，保護対象プログラムのソースコードを修正する必要はない．

我々は，制御 VM を通じて保護対象プログラムに関するデータを保護するために以下のコマンドを提供する．`vconf`，`vctl`，`vstart` は文献 35) にも記載しているため，本論文では簡略に述べる．

- `mkdummy`：これは制御 VM で管理する保護対象プログラムの保護対象ファイルの dummy ファイルを生成するためのコマンドである．保護対象プログラムを更新する場合には，dummy ファイルを再生成する．利用者はこのコマンドで生成される dummy ファイルを保護対象 VM 内に配置する．
- `vconf`：これは保護対象 OS 情報を Shadowall に登録するためのコマンドである．Shadowall では OS カーネルのバイナリイメージごとに保護対象 OS 情報を管理する．
- `vcntl`：これは `vconf` によって登録されている保護対象 OS 情報と保護対象 OS カーネルを関連付けるためのコマンドである．このコマンドが実行されると，保護対象 VM の外側からプロセスやシステムコールに関する制御ができるようになる．
- `vstart`：これは保護対象プログラムのデータ保護を開始するためのコマンドである．

図 2 では，保護対象 VM（VM ID:1）内の保護対象プログラム `target_prog` のデータ保護を開始するまでの流れが示されている．まず，保護対象プログラムに対応する dummy

© 2009 Information Processing Society of Japan

```
[cvm] $ mkdummy target_files.txt
[cvm] # vconf targetOS.img
            targetOS_info.txt syscall_info.txt targetOS_symbol.txt
[cvm] # vcntl 1 targetOS.img
[cvm] # vstart 1 target_prog policy.txt
```

図 2　利用例
Fig. 2　Usage example.

ファイルを生成するために，利用者は mkdummy を実行する．引数には保護対象 VM 内で保護するファイルに関する情報 target_files.txt を指定する．利用者は，target_files.txt に，保護対象 VM 内で保護する実行ファイルとセキュリティポリシで指定する保護対象プログラムに関連するファイルを記述する．次に vconf を用いて保護対象 OS に関する情報を制御デーモンに通知する．このコマンド引数には保護対象 OS のバイナリイメージ（targetOS.img）と保護対象 OS に関する 3 つの情報を与える．この 3 つの情報は，プロセス構造情報（targetOS_info.txt），システムコールに関する情報（syscall.txt），システムコールを捕捉するためのシンボル情報（targetOS_symbol.txt）からなる．さらに，vcntl により保護対象 VM の ID と保護対象 OS のバイナリイメージ（targetOS.img）を与え，保護対象 VM と保護対象 OS の情報を関連付ける．この例では vconf で登録された保護対象 OS 情報が VM ID 1 の中で稼働する OS 情報として用いられる．最後に，vstart により保護対象プログラムである target_prog，セキュリティポリシを記述したファイル（policy.txt）を指定し，保護対象プログラムのデータ保護の開始要求を発行する．保護対象プログラムの実行が開始されたときに，これらのコマンド実行で設定された情報に基づき，SW-core と制御プログラムにより保護対象プログラムのメモリ・仮想ディスク上のデータの保護が開始される．

3.4　データ保護機構

3.4.1　メモリ上のデータの保護

Linux や Windows を含むほとんどの commodity OS ではページング方式の仮想記憶により，異なる動作レベル（カーネルレベル・ユーザレベル）で同じページテーブルを共有する．このため，異なる動作レベルで保護対象プログラムの同じ仮想アドレスを操作した場合でも同じ物理アドレスへの操作となる．VMM を用いた場合，保護対象プログラムが操作する仮想アドレス（gVA）に対応する物理メモリ上のアドレス（hPA）への変換は VMM によって管理される．保護対象プログラムのユーザ空間のデータを保護するために，我々は動作レベルに基づき保護対象プログラムの物理ページを多重化するように保護対象 VM に関するメモリ管理操作を変更する（**図 3**）．1 つの仮想アドレスに対して動作レベルごとに異なる物理アドレスを対応付ける．この多重化処理の前後で，保護対象 OS カーネルが操作する物理アドレスは変わらない．動作レベルの切換えは，ページフォルト等の割込み・例外処理やシステムコール処理で発生する．

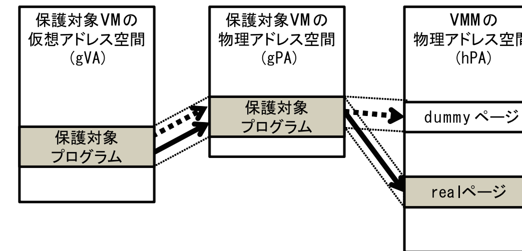

図 3　物理ページの多重化
Fig. 3　Physical page multiplexing.

保護対象プログラムがカーネルレベルで稼働しているときも含め，保護対象 OS カーネルから保護対象プログラムのメモリ上のデータを操作するときには，操作対象が dummy ページとなる．これにより，たとえ攻撃者が保護対象プログラムのメモリ領域の読み込みや書き込みを行ったとしても real ページは保護される．しかも，hPA への変換は VMM が制御しているため，保護対象 VM から real ページを操作できない．

システムコールにはファイルのパス名等ユーザ空間へのポインタを引数に含むものが存在する．保護対象 OS カーネルでこれらのシステムコールが正しく実行されるためには，real ページに含まれるデータを操作する必要がある．Shadowall では，システムコールのユーザ空間へのポインタ引数に対応する dummy ページと real ページの間で一時的にデータを共有することにより，システムコール処理の整合性を保つ．一時的に共有される dummy ページ内のデータの生存期間はシステムコール実行の開始から終了までの間である．ただし，保護対象ファイルに関するシステムコールがエミュレートされるときには，SW-core が real ページに対してのみエミュレート処理を行う．この場合，保護対象 OS カーネルによるシステムコール処理を経由しないため，dummy ページと real ページの間でデータを共有させる必要はない．

© 2009 Information Processing Society of Japan

既存のシステム[11),20),32)] のように，動作レベルごとに異なるメモリ上のデータを操作させるために暗号化技術を用いる手法では，暗号化・復号化にともなう実行時オーバヘッドが生じる．さらに，複数の仮想 CPU を利用する保護対象 VM 内で保護対象プログラムと保護対象外のプログラムを並列に実行させた場合に，保護対象プログラムに関するメモリ上のデータが並列に操作できないため，保護対象プログラムの実行性能が低下しうる．他方，Shadowall は保護すべき物理ページを複製し，動作レベルごとに異なる物理ページを操作するため，暗号化技術を用いた場合における実行時オーバヘッドは発生しない．

3.4.2 仮想ディスク上のデータ保護

Shadowall は，保護対象 VM とは異なる制御 VM で real ファイルを管理し，保護対象ファイルを操作したプログラムに基づいて操作対象のファイルを切り換える．これにより，保護対象プログラムが保護対象ファイルを操作したときのみ real ファイルにその操作が反映される．3.4.1 項のメモリ上のデータの保護の仕組みと組み合わせることで，メモリ上の保護対象ファイルの中身に対する不正操作も防げる．

保護対象プログラムによる保護対象ファイルに関するシステムコール手続きは保護対象 OS カーネル内の手続きを経由せず，代わりに SW-core と制御 VM 内の制御プログラムがエミュレートする．制御プログラムは read や write 等の保護対象ファイルの中身を操作するシステムコールをエミュレートする．stat や chown 等の保護対象ファイル情報に対するシステムコールは Shadowall ではエミュレートしない．これにより，エミュレートすべきファイル操作を軽減できる．また，chown で指定される保護対象 VM 内のユーザ ID と制御 VM に存在するユーザ ID の関係を考慮する必要もなくなる．Shadowall では，保護対象プログラムによるユーザ ID 等の変更を制御する場合には，文献 35) の仕組みを用いてそれらを制御することを想定している．また，制御対象外のプログラムにより dummy ファイルのユーザ ID 等が変更され，dummy ファイルの中身を操作できるようになったとしても，それが保護対象プログラムの実行には影響しない．保護対象プログラムは dummy ファイルの中身ではなく，real ファイルの中身に基づいて実行される．

保護対象 VM の外側で保護対象ファイルを管理する手段として，単純に NFS と VMM を組み合わせる手法（NFS-VMM）が考えられる．しかし，この手法では保護対象 VM 内の NFS クライアントによるデータ転送が保護対象 OS カーネル内の手続きを経由するため，保護対象 OS カーネルを奪取した攻撃者が保護対象ファイルを不正に操作できてしまう．他方，Shadowall では保護対象ファイルに関するメモリ上のデータを SW-core が直接操作するため，NFS-VMM におけるメモリ・ファイル間のデータ転送時の不正操作の問題が生じない．また，NFS-VMM では，制御 VM でのアクセス制御はユーザ ID に基づいた制御となり，プログラムに基づいたファイルへのアクセス制御ができない．Shadowall では保護対象 OS 情報を利用することで，保護対象 VM の外側からプログラムの識別ができる．

3.5 共有メモリ領域とネットワーク関連のデータ保護

Shadowall では保護対象プログラムの共有メモリ領域を利用するすべてのプログラムを保護対象プログラムとして指定することを想定している．また，libc のような共有ライブラリはそれらのメモリ上やファイル内のデータを隠蔽してしまうと多くのプログラムが正常に稼働しなくなってしまうため，共有ライブラリのメモリ上やファイル内のデータは Shadowall では隠蔽しない．ただし，保護対象プログラムに保護対象 VM 内の共有ライブラリではなく，制御 VM で管理する共有ライブラリを利用させることで，共有ライブラリのデータの改竄を防止することは可能である．

ネットワーク関連のデータ操作は保護対象 OS カーネル内の操作を経由するため，保護対象 OS カーネルを奪取した攻撃者がネットワーク関連のデータを不正に操作できてしまう．Shadowall においてネットワーク関連のデータを保護する場合には，保護対象プログラムが SSL を用いることを想定している．これにより保護対象のネットワーク関連のデータ自体は SSL による暗号化によって保護される．Shadowall では，SSL で用いる秘密鍵と証明書のファイルを保護対象ファイルとして指定し，これらに関連するメモリ上ディスク上のデータを保護する．なお，ウェブページのデータ等のネットワーク関連のデータは，外部に公開されるデータであるため，それらのデータの不正操作を防止することは Shadowall では対象としていない．

3.6 セキュリティポリシ

図 4 は Shadowall におけるセキュリティポリシ文法の一部を示している（すべてのセキュリティポリシ文法は付録 A.1 に記載）．Shadowall のセキュリティポリシでは，メモリ上のどのデータを保護するか，仮想ディスク上のファイルを保護対象とするかを指示する．

`executable` に続く部分では制御 VM で管理する保護対象プログラムの実行ファイルのパス名を *pathOutsideVM* に指定する．`shadowMemFile` に続く部分では，メモリ保護と保護対象ファイルに関して指示する．メモリ保護に関しては，すべての物理メモリ上のデータを多重化するか（`all`），または部分的に物理メモリ上のデータを多重化するか（`partially`）を指示する．

部分的に多重化する場合（PartSpec）には，OS レベルに関する指示（KernSeg）や mmap 領域に関する指示（`mmapRegion`），ELF 実行形式のレベルでの指示（ELFSec）が可能で

© 2009 Information Processing Society of Japan

PolicyFile	→	`executable` : *pathOutsideVM*
		`shadowMemoryFile` : ShadowMemFile
ShadowMemFile	→	`all` ShadowFileSpec∗
		| `partially` PartSpec+ ShadowFileSpec∗
PartSpec	→	`kernelSeg` : KernSeg+ | `mmapRegion` | `ELFSec` : *secName*+
		| `readOnlyRegions` : RORegion+
KernSeg	→	`text` | `data` | `brk` | `stack(DOWN|UP,` *size*`)` | `env` | `arg`
RORegion	→	`text` | `ELFSec` : *secName*+ | `mmapRegion`
ShadowFileSpec	→	`shadowFile` : FileSpec+
FileSpec	→	`pathName(`*insideVM*, *outsideVM*`)` <Permission+>
Permission	→	`read` | `write` | `create` | `append`

図 4 ポリシ文法（抜粋）
Fig. 4 Security policy syntax (extracted).

ある．OS レベルに関する指示では，保護対象プログラムのコード領域（`text`），静的・動的データ領域（`data`，`brk`），スタック，環境変数領域およびプログラム引数領域（`stack`，`arg`，`env`）を指定することができる．さらに，メモリ上のデータ領域の読み込みのみを許可する指示（RORegion）も可能である．Shadowall では RORegion のメモリ領域は多重化せず，保護対象 VM からは読み込み専用に設定される．

部分的な指定（`partially`）の利用例を以下に述べる．まず，パスワードが含まれるメモリ領域や認証鍵を含んでいるファイルがメモリにマップされた領域を部分的に保護するメモリ領域として指定する場合があげられる．また，保護対象プログラムの開発者がソースコードで保護すべき機密データを含む ELF 実行形式の特別なセクションを定義し，セキュリティポリシでその ELF 実行形式の特別なセクションを保護対象領域として部分的に指定するような場合があげられる．ただし，これらのように部分的な指定を用いた場合，指定したメモリ領域以外が改竄され，保護対象プログラムが異常な振舞いをする可能性がありうる．これに対しては，我々は保護対象 VM の外側で稼働させるセキュリティシステム[17),35)]と組み合わせることでこの攻撃を緩和させられると考えている．さらに，部分的なメモリ多重化は必ずしもすべてのメモリ領域を多重化する必要がない場合にも，多重化するメモリ領域を削減するために有用である．すでに外部にデータが公開されているプログラムは，コード領域や読み込み専用のデータ領域等は改竄のみを防げればよいため，これらの領域は

```
executable : ./shadow/master_prog
  shadowMemoryFile : partially
     kernelSeg : data,brk,stack(DOWN,1GB),env,arg,mmapRegion
     readOnlyRegions : text
  shadowFile :
     pathName(/home/A/in_file1.cfg, ./out_file1.cfg) ⟨read⟩
```

図 5 ポリシの記述例
Fig. 5 Sample security policy.

読み込みのみを許可するようにセキュリティポリシで指定する．残りのメモリ領域に対して部分的な指定を用いる．

ShadowFileSpec では，保護対象 VM 固有の保護対象ファイルの保護に関して指示する．*insideVM* に dummy ファイルのパス名を，*outsideVM* には *insideVM* に対応する real ファイルのパス名を指定する．制御プログラムの real ファイルに対する操作権限を Permission で指定する．

図 5 では，保護対象プログラム `master_prog` に対するセキュリティポリシの記述例が示されている．このポリシによって，`master_prog` のメモリ上のデータに関して，コード領域（`text`）を含む物理メモリ領域が読み込み専用となり，他の物理メモリ領域は多重化される．保護対象ファイルとして，設定ファイル（`out_file1.cfg`）が読み込み専用となる．

4. 実　　装

我々は，VMM として準仮想化を用いた Xen 3.0.3，保護対象 OS として Linux 2.6.16 を用いて，CPU アーキテクチャAMD64 上で Shadowall の設計および実装を行った．Shadowall を利用する際，保護対象 OS カーネルのソースコードを修正する必要はない．さらに，保護対象 VM 内に保護対象プログラムのための Shadowall 専用プログラムを追加する必要もない．

4.1 Dummy ファイルの生成

Shadowall では，保護対象ファイルの生成・更新ごとに，dummy ファイルを生成・更新する．保護対象プログラムは保護対象 VM 内で dummy 実行ファイルを用いて起動する．Linux OS カーネルで ELF 形式の実行ファイルを処理するときには，ELF ヘッダ，ELF のプログラムヘッダおよび `.interp` セクションに関する情報が必要となる．このため，Shadowall

© 2009 Information Processing Society of Japan

では dummy 実行ファイル内のデータにこれらの情報を含める．その他の保護対象ファイルの dummy ファイルの中身はすべてゼロクリアする．ただし，保護対象外のプログラムも利用する可能性のある libc のような共有ライブラリの中身はゼロクリアしない．Shadowall では，保護対象ファイルの real ファイルを入力として受け取り，対応する dummy ファイルを生成するために，3.3 節で述べたコマンド `mkdummy` を提供している．

4.2 動作レベルの切換えの捕捉

動作レベルの違いに基づいて多重化された保護対象プログラムが操作する物理メモリ領域を切り換えるため，VMM が割込み・例外処理とシステムコール処理の開始・終了時を捕捉する必要がある．VMM 上で稼働する VM の割込み・例外処理を制御するため，VMM は割込み・例外処理の開始を捕捉可能である．準仮想化を利用した Xen の CPU アーキテクチャAMD64 版（Xen-AMD64）では，システムコールの実行は SYSCALL 命令で開始されるため，VMM がシステムコール開始時の捕捉もできる．さらに，Xen-AMD64 では割込み・例外処理の終了時には Hypercall を利用するため，VMM は割込み・例外処理の終了時を捕捉することもできる．Shadowall ではシステムコール処理の終了直後の保護対象プログラムのレジスタやメモリ上の実行状態から保護対象プログラムの実行状態を取得することが必要である．このため，我々はシステムコール処理の終了時を捕捉するために保護対象 OS カーネルイメージのバイナリ書き換え[35] を用いている．

4.3 物理ページの多重化

4.3.1 メモリ領域情報の管理

保護対象プログラムのユーザ空間を保護するために，SW-core は保護すべきメモリマップ領域と物理メモリ領域に関する 2 つの情報を管理する．メモリマップ領域に関する情報（メモリマップ情報）はプロセスごとに管理し，各プロセスの現在の保護すべき仮想アドレスの範囲に関する情報が含まれる．この情報は，直感的には Linux OS カーネルにおける `vm_area_struct` に相当する．一方，保護すべき物理メモリ領域に関する情報（物理メモリ情報）は VM ごとに管理される．物理メモリ情報には，ユーザレベルの稼働中に参照する物理アドレス（real アドレス）やそれに対応するカーネルレベルの稼働中に参照する物理アドレス（dummy アドレス）等の情報がページ単位で含まれる．さらに物理メモリ情報には real アドレスごとに保護対象 VM の制御を開始したときからの参照数も含まれる．

4.3.2 ページテーブルの更新

物理ページを多重化するため，Shadowall では動作レベルごとに異なるページテーブルを設定する．Xen-AMD64 では，動作レベルごとに CR3 レジスタに異なる最上位のページテーブルが設定される．このため，Shadowall では Xen-AMD64 の実装を拡張して保護対象プログラムの物理ページを多重化する．

多重化すべきメモリマップ領域内でページフォルトが発生したときに SW-core が物理ページを多重化する．このとき，SW-core は保護対象 OS カーネルによるページフォルト処理と整合性を保つように多重化処理を行う必要がある．VMM がページフォルト発生前後のページテーブルエントリ（PTE）の遷移に基づいて行う対応処理が**表 1** に示されている．

dummy ページが存在しない場合，最初に SW-core がページフォルトを捕捉した時点では保護対象 OS カーネルによるページフォルト処理前であるため，物理ページ（dummy ページ）は保護対象 OS カーネルによってまだ割り当てられていない．この時点で，SW-core は CR3 とページフォルト発生時のインストラクションポインタ（IP）を保存し，保護対象 VM を再開させる．次に，SW-core が保護対象 OS カーネルによるページフォルト処理後のユーザレベルへの遷移を捕捉したときには dummy ページが割り当てられている（PTE $\neq$ 0）．この例外処理の捕捉時に，保存した CR3 と IP に基づき多重化処理を行うかを決定する，多重化処理が必要である場合には，SW-core が dummy ページに対応する real ページを割り当てる．

Copy-on-Write（COW）の場合，PTE の物理ページベースアドレス（PPBA）がページフォルト前後で同じ場合，real ページに対応する PTE を変更する．他方，PPBA がページフォルト前後で異なる場合には，dummy ページに対応する real ページを割り当てた後，元の real ページからページの中身を新たに確保した real ページにコピーする．そして，real ページに対応する PTE を変更する．

dummy ページのページインの場合，dummy ページに対応するページアウトされていた real ページを PTE に設定する．ページアウトされる dummy ページは VMM が管理し，ページテーブルの更新時にページアウトに関する情報が追加される．保護対象 OS カーネルによるページイン・ページアウト処理は dummy ページに対する処理となるため，攻撃者がこれらの処理中に real ページ内のデータを不正操作することはできない．

準仮想化を利用した Xen（PV-Xen）では最下位のページテーブルは読み込み専用となっており，PTE は，保護対象 OS カーネルによる Hypercall 経由または VMM によるページフォルト処理で更新される．**表 2** で示されているように，SW-core がこれらの処理を捕捉したときに，対応する real PTE を更新する．dummy ページがページアウトする場合には，VMM が PTE と対応する real ページを管理下に置く．また，dummy ページに対応する PTE がゼロクリアされる場合，real ページの参照数を 1 減らす．そして，参照数が 0 にな

© 2009 Information Processing Society of Japan

表 1 ページテーブルエントリの遷移に基づくページフォルト処理
Table 1 Page fault handling for each page table entry transition.

ページフォルト処理前			ページフォルト処理後			
PPBA	P ビット	R/W ビット	PPBA	P ビット	R/W ビット	対応処理
0	0	-	$\neq 0$	1	-	新しいページの割当
$\neq 0$	1	0	$\neq 0$	1	1	Copy-on-Write
$\neq 0$	0	-	$\neq 0$	1	-	ページイン

表 2 ページテーブルエントリの遷移に基づくページテーブルエントリの更新
Table 2 Page table entry update for each page table entry transition.

更新前			更新後			
PPBA	P ビット	R/W ビット	PPBA	P ビット	R/W ビット	対応処理
$\neq 0$	1	0/1	$\neq 0$	1	1/0	R/W ビットの変更
$\neq 0$	1	-	$\neq 0$	0	-	ページアウト
$\neq 0$	0	-	0	0	0	ページの解放*1

PPBA：物理ページベースアドレス，P：Present，R/W：Read/Write，-：0 と 1 のどちらでもよい．

る場合には real ページ等を解放する．

セキュリティポリシの指示によりメモリ領域を部分的に多重化させる場合には，SW-core が多重化する対象ではないページでも，R/W ビットが立っている書き込み可能なページを多重化する．これは保護対象のデータを含むページ内のデータを保護対象ではないページに漏洩させる攻撃を防ぐために必要となる．

4.4 保護対象ファイル操作のエミュレート

Shadowall では，保護対象プログラムによる保護対象ファイルに関するシステムコール手続きは保護対象 OS カーネル内の手続きを経由せず，代わりに SW-core と制御 VM 内の制御プログラムがエミュレートする．制御プログラムは保護対象ファイルに対する read や write 等のファイルの中身を操作するシステムコールをエミュレートする．保護対象ファイルに対する操作であるかの判断にはシステムコール捕捉時に取得するファイル記述子を用いる．このため，open 等の捕捉時に保護対象のファイル記述子を SW-core と制御プログラムで保持しておく必要がある．また，エミュレートされるシステムコールは保護対象 VM 内でも dummy 保護対象ファイルに対して実行される．SW-core と制御プログラムの間のイベント通信には Xen が提供している event channel と制御プログラムと SW-core 間の共有メモリを利用している．

*1 real ページの参照数が 0 になる場合．

4.5 システムコール処理

4.5.1 ユーザアドレス空間へのポインタ引数に関する操作

保護対象 OS カーネルによるシステムコール処理時の整合性を保つために，動作レベルごとに異なる物理ページを操作するシステムコールのユーザ空間へのポインタ引数に対応する必要がある．Shadowall では，dummy メモリ領域と real メモリ領域の間で，システムコール実行の開始から終了まで一時的にデータを共有することにより，システムコール処理時の整合性を保つ．3.3 節で述べた `vconf` の実行時に，保護対象 OS カーネルにおける各システムコールのポインタ引数に関する情報が SW-core に通知される．このシステムコールのポインタ引数に関する情報には，システムコール番号，ポインタ引数番号，ポインタ引数の大きさに関する情報が含まれる．各ポインタ引数の大きさは保護対象 OS カーネルに依存する．このため，Shadowall ではカーネルイメージのコンパイル時に各ポインタ引数の大きさを自動生成するツールを提供している．

SW-core はシステムコールのポインタ引数に関する情報を用いて，システムコールの開始・終了時にデータ共有処理を行う．SW-core はシステムコール実行開始の捕捉時に，まず，ポインタ引数のユーザ空間を指す dummy 物理メモリ領域が割り当てられているかを検査する．もし，dummy 物理ページが割り当てられていない場合には，そのポインタが指す仮想アドレスでページフォルトが発生したように見せかけ，保護対象 VM を再開させる．これにより，保護対象 OS カーネルのページフォルトハンドラによって dummy メモリ領域が割り当てられる．システムコール実行の開始時のポインタ引数に関する操作では，real メモリ領域から dummy メモリ領域へデータが受け渡される．一方，システムコール実行終了のポインタ引数に関する操作では，dummy メモリ領域から real メモリ領域へデータが受け渡される．システムコール実行の終了時には，最後に real メモリ領域とデータを共有するために利用した dummy メモリ領域内のデータをゼロクリアする．

4.5.2 保護対象プログラムの実行制御の開始・終了

execve で保護対象プログラムの実行が開始されたときに，保護対象プログラムに関するメモリマップ情報を初期化する．メモリマップ情報は保護対象プログラムのプロセス管理構造体 `task_struct` から走査可能なメモリ管理構造体 `mm_struct` のメンバ `start_code` や `start_data` 等に基づいて初期化される．このとき，初期化されるメモリマップ情報には，制御 VM 内で読み込まれる保護対象プログラムの実行ファイルの中身も追加される．実行ファイルのデータに対応する物理ページがすでに保護対象 OS カーネルによって割り当てられている場合には，SW-core が対応する物理ページに実行ファイルの中身をコピーする．

© 2009 Information Processing Society of Japan

セキュリティポリシで多重化するように指示されている場合には，SW-core がこの時点で物理ページの多重化も行う．

exit_group や exit が実行され，保護対象のプロセスが実行を終了するときには，メモリマップ領域や物理メモリ情報に関する情報が削除される．

4.5.3　ヒープ領域・メモリマップ領域に関する処理

mmap や brk 等を SW-core が捕捉したときに，保護対象プログラムに関するメモリマップ情報が更新される．mmap や mprotect 等の捕捉時には読み込み・書き込み権限もメモリマップ情報に追加される．さらに，mmap の引数で指定されたファイル記述子が保護対象ファイルのファイル記述子であった場合には，その情報もメモリマップ情報に追加される．SW-core はメモリマップ情報に基づき，ページフォルトやページテーブルエントリの更新時に real メモリ領域に関する対応処理を行う．保護対象ファイルがメモリ上にマップされた領域で，物理ページが確保されていないためにページフォルトが発生した場合には，SW-core と制御プログラムにより real メモリ領域に関する対応処理が行われる．さらに，mmap の引数で PROT_WRITE と MAP_SHARED が指定されたメモリ領域に関しては，munmap や close の捕捉時に real メモリ領域に対する更新を，制御プログラムが制御 VM 内の保護対象ファイルの中身に反映させる．

4.5.4　シグナル処理

Linux におけるシグナルの受信処理時にはカーネルレベルとユーザレベルの間で動作モードが変更される．シグナル受信処理時には，Linux OS カーネルはシグナルハンドラの引数やリターンアドレス，その時点のカーネルの実行状態等を保存するために，ユーザレベルスタックを用いる．ユーザレベルのスタック領域が多重化されている場合，システムコールのユーザ空間を指すポインタに関する処理と同様に，シグナルハンドラの実行の捕捉と real スタック領域へのシグナルハンドラで利用する情報の受け渡しが必要となる．Shadowall では保護対象プログラムがユーザ定義シグナル関数を設定する sigaction や signal 等の捕捉時に，バイナリ書き換え[35] を用いて，シグナルハンドラの実行を捕捉できるように設定する．シグナルハンドラの実行の捕捉時に必要なデータを real メモリ領域内のスタック領域へコピーする．

4.5.5　プロセス生成

fork により子プロセスが生成される場合，SW-core は，親プロセスの保護対象メモリに関するメモリマップ情報を複製する．また，fork の終了の時点での子プロセスの各ページテーブルエントリは dummy 物理ページに対する設定となっている．このため，ユーザレベル時に対応する real 物理ページを操作するように，dummy 物理ページが設定されている各ページテーブルエントリの変更も行う．さらに，fork 時に開かれている保護対象ファイルに関連するファイル記述子に関する情報も複製される．ユーザ定義のシグナルハンドラ情報も複製される．

スレッド（light-weight process）の生成等で利用される clone が実行され，CLONE_VM が引数に含まれる場合には，メモリマップ情報を生成元のプロセスと共有する．また，CLONE_FILES が指定されていた場合には生成元のプロセスと保護対象ファイルに関するファイル記述子を共有する．さらに，CLONE_SIGHAND が指定されていた場合には生成元のプロセスとユーザ定義のシグナルハンドラ情報も共有する．

4.5.6　プロセス間通信

Linux におけるパイプやメッセージキューを用いたプロセス間通信では，OS カーネルが通信データを保持する．このため，攻撃者がパイプやメッセージキューで利用する通信データを不正操作することが可能となる．Shadowall では，通信データの送受信前後で，通信データを暗号化・復号化することにより通信データを保護する．パイプを用いたプロセス間通信では，パイプによって生成されるファイル記述子に基づいて暗号化・復号化処理を行う．メッセージキューを用いたプロセス間通信では，msgsnd や msgrcv が実行されたときに暗号化・復号化処理を行う．

5．評　　価

Shadowall を評価するため，保護対象プログラムのメモリ・仮想ディスク上のデータの保護の確認と実行時のオーバヘッドの計測を行った．実験環境は CPU Dual-Core AMD Opteron 2.8 GHz が 2 つ，8 GB メモリ，1 Gbps NIC である．制御 VM と保護対象 VM は各々4 つの仮想 CPU を割り当て，メモリサイズは各々1 GB に設定した．

5.1　メモリ・仮想ディスク上データの保護の確認

保護対象プログラムとしてウェブサーバ thttpd とアンチウィルスツール ClamAV を用いた．

まず，thttpd の chroot jail 機能を利用して thttpd のルートディレクトリを `/var/www` に設定して thttpd を保護対象 VM 内で稼働させた．これにより thttpd は `/usr/bin/perl` 等へのアクセスが制限され，CGI プログラムを実行できなくなる．次に，我々は攻撃者を模擬し，thttpd のメモリ上と仮想ディスク上のデータを改竄することにより，thttpd が CGI プログラムを実行できるようにした．これを実現するために，我々は thttpd を再起動させ，

© 2009 Information Processing Society of Japan

メモリ上の変数 `do_chroot` を改竄して chroot jail 機能を無効化した．これはデバッガ GDB を用いて，thttpd の初期化処理中に `do_chroot` がセットされた後で thttpd を停止させ，`do_chroot` をクリアさせることで実現した．これらの改竄により，thttpd の設定ファイルには chroot jail 機能が有効になるような記述があるにもかかわらず，thttpd で CGI プログラムを実行させせることができた．さらに，ログファイル `/var/log/syslog` から thttpd の再起動のログと chroot jail 機能が無効であることを示すログを削除した．

次に，Shadowall 上で thttpd の物理メモリ上のデータが多重化されるようなポリシを用いて thttpd のデータを保護した．さらに syslogd の物理メモリ上のデータと `/var/log/syslog` が保護されるようなポリシも用いて syslogd のデータを保護した．この状態で同様の不正操作を試みたが，CGI プログラムを実行することはできなかった．thttpd のコード領域はゼロクリアされているため，GDB が一時停止のためのブレークポイントをメモリ上に設定できなかった．また，`/var/log/syslog` は保護対象 VM の外側で管理されているため改竄することはできなかった．

次に，制御 VM で ClamAV が提供しているウィルス検査プログラム clamscan を用いてウィルスファイル `mw.exe` を検査した．まず，`mw.exe` からウィルスシグネチャを含むデータベースファイル `mw.hdb` を生成した．その後，`mw.hdb` を用いて clamscan で `mw.exe` を検査し，`mw.exe` がウィルスを含むことを標準出力に表示した．このとき，我々は 2 通りの方法で `mw.exe` がウィルスを含まないという誤った表示をさせた．1 つの方法は GDB を用いて，メモリ上のウィルス検査結果を表す，関数 `cl_scandesc` の返り値を改竄した．別の方法ではデータベースファイル内のウィルスシグネチャを改竄した．これらにより，誤った実行結果が表示された．

次に，clamscan のメモリ上のデータが多重化され，`mw.hdb` を保護対象ファイルとするようなセキュリティポリシを用いてウィルス検査を実行した．thttpd の場合と同様に，メモリ上の `cl_scandesc` の返り値や `mw.hdb` の中身を改竄することはできなかった．このため，clamscan による検査結果は `mw.exe` がウィルスを含んでいると正しく表示された．

5.2　実行時オーバヘッドの測定

5.2.1　マイクロベンチマーク

Shadowall，Xen 上と物理計算機上（Linux）で以下のベンチマークプログラムを各々1,000 回繰り返し，その実行時間を測定した．Shadowall の場合では，ベンチマークプログラムを保護対象とする場合（target）と保護対象としない場合（non-target）の実行時間を測定した．Xen の場合と比べ，non-target の場合にはシステムコール捕捉や保護対象のプロセスの識別等の操作が加わる．Linux では物理メモリサイズを 1 GB に設定した．

- getpid（getpid）：基本的なシステムコール捕捉に要する時間を計測する．
- mmap，munmap（mmap，mmapW）：4 KB の無名メモリマップ領域を確保・解放する処理に要する時間を計測する．mmap ではメモリマップ情報が追加・削除される．mmapW ではメモリマップ領域へデータが書き込まれたときにページフォルトが発生し，さらに物理メモリ情報の追加・削除が必要となる．
- open，close（open）：ホームディレクトリ下のファイル test.txt を保護対象ファイルとして管理する処理に要する時間を計測する．
- open，read，close（readIn，readOut）：保護対象 VM 内（readIn）と制御 VM 内（readOut）の各々の test.txt から 1 KB のデータを読み込む処理に要する時間を計測する．保護対象 VM 内の場合には動作レベルごとの読み込み用のバッファ領域間でデータをコピーする必要がある．
- open，write，close（writeIn，readOut）：保護対象 VM 内（writeIn）と制御 VM 内（writeOut）の各々の test.txt へ 1 KB のデータを書き込む処理に要する時間を計測する．
- fork，wait4，exit_group（fork）：生成された子プロセスを保護対象に追加・削除する処理や多重化に関連する処理が必要となる．

表 3 の実行結果は一連のマイクロベンチマークの 1 回の実行に要する時間を表している．readOut と writeOut は保護対象 VM の外側で保護対象ファイルに関する操作をエミュレートした場合の測定であるため，表 3 中の Shadowall (target) のみ測定している（表 3 中の ‘-’ は測定値なしを表す）．Xen と比べて，保護対象ファイルのエミュレーションに関する結果を除き，保護対象とした場合に対するオーバヘッドは約 1.8 倍から 4.5 倍，保護対象としない場合に対するオーバヘッドは，約 1.1 倍から 3.2 倍であった．保護対象ファイルのエミュレーションに関する処理では約 16 倍から約 22 倍であった．他方，Linux と比べて，getpid と保護対象ファイルのエミュレーションに関する結果を除き，保護対象とした場合と保護対象としない場合に対するオーバヘッドは各々，約 6.6 倍から 9.3 倍，約 2.7 倍から 5.6 倍であった．getpid と保護対象ファイルのエミュレーションに関する処理では約 25 倍から約 39 倍であった．Shadowall によって生じる実行時のオーバヘッドはマイクロベンチマークプログラムに対して大きかったが，より規模の大きなアプリケーションプログラムを Shadowall へ適用した場合には，Shadowall によって生じるオーバヘッドは緩和されると考えている．

© 2009 Information Processing Society of Japan

表 3　マイクロベンチマークの結果
Table 3　Microbenchmark results.

	getpid	mmap/mmapW	open	readIn/readOut	writeIn/writeOut	fork
Shadowall (target)	2.36	6.46 / 18.6	81.1	25.9 / 101	27.7 / 107	310
Shadowall (non-target)	1.65	4.46 / 11.6	6.10	9.53 / -	11.6 / -	182
Xen	0.52	2.20 / 8.06	3.63	5.95 / -	6.89 / -	172
Linux	0.06	0.80 / 2.40	2.27	3.00 / -	4.21 / -	33.4

［マイクロ秒］

表 4　アプリケーションベンチマークの結果
Table 4　Application benchmark results.

	thttpd		clamscan	
	FS-1KB［ミリ秒/要求］	FS-100KB［ミリ秒/要求］	VDB-inVM［ミリ秒］	VDB-outVM［ミリ秒］
Shadowall	0.8	9.4	173	181
Xen	0.7	9.3	168	-
Linux	0.5	9.1	105	-

5.2.2　アプリケーションベンチマーク

ここでは保護対象プログラムとして thttpd と ClamAV を用いた．この実験のセキュリティポリシではメモリ上のデータをすべて多重化（`all`）し，保護対象ファイルに設定ファイルとログファイルを指示した．比較のため，Xen 上と物理計算機上（Linux）で同様の測定を行った．Linux では物理メモリサイズを 1 GB に設定した．

thttpd に対しては，httperf を用いてウェブサービススループットを計測した．httperf を実行した物理計算機は，Pentium 4 3.0 GHz（HT 有効），1 GB メモリ，1 Gbps NIC であり，保護対象 VM の物理計算機とは同一 LAN に設置した．ここでは httperf から接続数が 128 の場合に対する静的コンテンツ（ファイルサイズ（FS）：1 KB，100 KB）の要求を発行した．

ClamAV に対しては，clamscan を用いて 15 のファイル（ウィルスファイルは 5 つ）のウィルス検査に要する時間を計測した．ここでは，5 つのウィルスファイルからウィルスデータベース（VDB）を生成し，保護対象 VM 内（VDB-inVM）と保護対象 VM の外側（VDB-outVM）にある各々の場合に対して実行時間を測定した．VDB-outVM の場合には保護対象ファイルの中身に関するシステムコール処理がエミュレートされるが，VDB-inVM の場合にはシステムコール処理がエミュレートされない．

表 4 に実験結果が示されている．VDB-outVM は保護対象 VM の外側で保護対象ファイルに関する操作をエミュレートした場合の測定であるため，Shadowall の場合のみ測定している（表 4 中の ‘-’ は測定値なしを表す）．表 4 より，ウェブサービススループットでは，Xen と比較して約 1%から 14%，Linux と比較して約 3%から約 60%のオーバヘッドが生じた．ウィルス検査では，Xen と比較して，保護対象 VM 内で VDB を管理する場合には約 3%，保護対象 VM の外側で VDB を管理する場合には約 8%のオーバヘッドが生じた．他方，Linux と比較した場合，保護対象 VM 内で VDB を管理する場合には約 65%，保護対象 VM の外側で VDB を管理する場合には約 72%のオーバヘッドが生じた．Shadowall を利用することで生じるオーバヘッドよりも，VMM を利用することで生じるオーバヘッドの方が大きかった．

これらの実験結果を通じて，Linux の場合と比べ，Shadowall を利用することである程度のオーバヘッドは生じるが，保護対象プログラムに関するメモリ・ディスク上のデータを保護することができると考えている．

6. 関 連 研 究

6.1　保護対象 VM の外からのアプリケーションのデータの保護

VMM を用いた物理ページの多重化により保護対象プログラムのデータを保護するシステムがこれまで提案されている[11),26),32)]．これらのシステムは保護対象プログラムを操作するために，保護対象 VM 内で専用のユーザレベルプログラムや操作コマンドが必要である．一方，Shadowall は保護対象 VM の外側から，セキュリティポリシを用いて保護対象プログラムを制御する．Overshadow[11)] と SP[3 32)] はカーネルレベルの動作時に保護対象プログラムの物理ページを暗号化することにより物理ページを多重化しているように見せるが，Shadowall は保護対象プログラムの物理ページを複製している．また，これらのシステムはメモリ上の保護領域が保護対象プログラム全体であるのに対し，Shadowall はセキュリティポリシを用いて保護領域を部分的に指示することも可能である．文献 26) のシステ

© 2009 Information Processing Society of Japan

ムは，アドレス参照が命令参照であるかデータ参照であるかによって参照先が異なるように物理アドレスを複製する．他方，Shadowall は動作レベルの違いに基づいて物理ページを複製する．Overshadow[11] による保護対象ファイルの保護はメモリ上のデータと同様，暗号化によって実現されている．Shadowall は保護対象 VM の外側で保護対象ファイルを管理しているため，Overshadow と異なり，保護対象ファイルの改竄も防ぐことができる．SP[3] や文献 26) のシステムは保護対象ファイルは保護しない．

XOM[20] は，OS カーネルを信頼せずに，保護対象プログラムのコード領域やデータ領域に関するデータの漏洩や改竄を防ぐための専用の CPU アーキテクチャである．XOM 専用命令と暗号化技術により，保護対象プログラムの稼働中とその他で物理メモリの中身が異なるように見せる．XOM によって保護対象プログラムを保護するためには，XOM 専用の OS カーネル（XOMOS[19]）を利用する必要がある．他方，Shadowall では既存の commodity OS である Linux に適用できる．

保護対象 VM とは異なる保護ドメイン内で保護対象プログラムを稼働させるシステムが提案されている[12),13),15),28]．これらのシステムとは異なり，Shadowall は保護対象プログラムが保護対象 VM 内で稼働する．Proxos[28] では，保護対象プログラムが信頼すべきシステムコールを制御 VM 内で実行し，他の信頼できないシステムコールは保護対象 VM 内で実行させる．Proxos は，保護対象 OS カーネルに加え，保護対象プログラムのソースコードを修正する必要があるが，Shadowall はそれらのソースコードを修正する必要がない．さらに，Proxos のセキュリティポリシでは信頼すべきシステムコールの分類を指示するのに対し，Shadowall は保護対象のメモリ領域とファイルを指示する．Nizza[15] はマイクロカーネルを利用し，保護対象プログラムの処理の中で鍵認証のような信頼すべき処理のみを異なる保護ドメイン内で実行させ，保護対象プログラムの TCB を削減するシステムである．Proxos 同様，Nizza を利用する場合には保護対象プログラムや保護対象 VM 内で保護対象プログラムと連携するプログラムのソースコードを修正する必要がある．

保護対象 VM 内で利用するアクセス制御を保護対象 VM の外で行い，共有ライブラリ等の sensitive ファイルを保護するシステムがこれまで提案されている[33),34]．これらのシステムはファイルシステムレベルでアクセス制御により sensitive ファイルを保護するのに対し，Shadowall はシステムコールレベルで保護対象プログラムに関連するファイルの中身を保護する．また，これらのシステムはメモリ・ファイル間のデータ転送時に保護対象 OS カーネル内の手続きを利用するため，保護対象 OS カーネルが乗っ取られた場合これらのメモリ・ファイル上の保護対象ファイルに関するデータが改竄されうる．一方，Shadowall は SW-core が保護対象プログラムが利用するメモリ領域を直接操作してメモリ・ファイル間のデータ転送を行うため改竄を防止できる．

6.2 OS・アプリケーションレベルでのアプリケーションのデータの保護

OS レベルで，信頼できないアプリケーションの実行を制限するためのセキュリティシステムが数多く提案されている．PeaPod[24] は OS レベルの仮想化を利用し，VM 単位の隔離よりも消費計算資源を抑えられるプロセス単位の保護ドメインによる隔離と保護ドメイン内のアプリケーションコンポーネント間のアクセス制御により，信頼できないアプリケーションの振舞いを制御するシステムである．PeaPod はアプリケーションに関するシステムコール捕捉と chroot システムコールによる名前空間の隔離によってこれを実現している．Janus[14] や Systrace[25] では，信頼のおけないプログラムが発行するシステムコールを，セキュリティポリシに基づき制御するサンドボックスシステムである．Solitude[16] は，セキュリティポリシに基づいて，ファイルシステムレベルでの信頼のおけないプログラムの名前空間の分離と障害からの復旧処理を行うシステムである．これらのシステムは，セキュリティポリシで指示されていないプログラムの振舞いを制御できない．SELinux[21] や LIDS[3] は強制アクセス制御に基づき，実行環境内のファイル操作を制御するセキュア OS である．これらの OS レベルでの保護対象プログラムに関連するデータ保護は，制御プログラムが OS カーネルを含め他のプログラムと同一実行空間で稼働する．このため，OS カーネルまたは管理者権限で稼働するプログラムが奪取された場合，攻撃者が制御プログラムを無効化することが可能になってしまう．

7. まとめと今後の課題

本論文では，VM の外側から内部で稼働する保護対象プログラムに関するデータ操作を制御し，メモリ・仮想ディスク上の保護対象プログラムに関連するデータを保護するアーキテクチャShadowall を提案した．Shadowall を利用することで，たとえ保護対象 OS カーネルを含む保護対象 VM 内のプログラムが奪取された場合でも，保護対象プログラムに関するメモリ・仮想ディスク上のデータの漏洩や改竄を妨げられる．メモリ上のデータを保護するために，ユーザレベルとカーネルレベルの動作レベルごとに保護対象プログラムが異なるユーザアドレス空間を操作するように，VMM 内コンポーネント SW-core が 1 つの仮想アドレスに対応する物理ページを多重化する．この SW-core によるアドレス変換操作は VMM 層で行われるため，保護対象 VM 内の OS カーネルが保護対象プログラムのメモリ上のデータを不正操作できない．仮想ディスク上のデータを保護するために，実行ファイル

© 2009 Information Processing Society of Japan

や設定ファイル等の保護対象ファイルを保護対象 VM とは異なる制御 VM で管理する．保護対象プログラムが read，write のような保護対象ファイルの中身を操作するシステムコールを発行したときに SW-core が捕捉し，SW-core と制御 VM によりそのファイル操作をエミュレートする．メモリ上のデータや保護すべきファイルの指定は，利用者がセキュリティポリシによって指示する．我々は VMM として準仮想化を利用した Xen を，保護対象 OS として Linux を用いて，AMD64 アーキテクチャ上で Shadowall の設計，実装および評価を行った．

今後の課題としては，カーネルレベルルートキット等により，カーネル空間の保護対象プログラムに関連するデータが改竄された場合への対応があげられる．また，他のプログラムのメモリ上の実行状態を解析・制御する既存のプログラムとの互換性を保つことがあげられる．このことは，ptrace を利用するデバッガやセキュリティシステムに対しては，ptrace で有効なメモリ上のデータ操作を許可するよう，セキュリティポリシで指示できるようにしたい．ptrace 実行時には，保護対象ファイルのエミュレーションと同様に，SW-core が ptrace の操作をエミュレートすることで特定の保護対象プログラムのみ他のプログラムのメモリに関する実行状態を操作できるようにすることができると考えている．さらに，メモリ・仮想ディスク上のデータの保護に加え，保護対象プログラムが動作レベルをカーネルレベルへ切り換える際ときのレジスタの値の保護についても検討したい．

謝辞　本研究に関して適切な助言をくださった前田助教をはじめとした米澤研究室の方々，ならびに本論文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) Adobe Flash player code execution vulnerability (VU#395473). http://www.kb.cert.org/vuls/id/395473
2) Debian and Ubuntu OpenSSL packages contain a predictable random number generator (VU#925211). http://www.kb.cert.org/vuls/id/925211
3) Linux Intrusion Detection System. http://www.lids.org/
4) Linux Kernel "exit_notify()" CAP_KILL Verification Local Privilege Escalation Vulnerability. http://securityfocus.com/bid/34405/
5) Microsoft Security Bulletin MS07-066. http://www.microsoft.com/technet/security/bulletin/MS07-066.mspx
6) MIT Kerberos 5 RPC Library Remote Code Execution Vulnerability (CVE-2006-6143). http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6143
7) Remote Code Execution in Samba's nmbd (CVE-2007-5398). http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5398
8) 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. 19th ACM Symposium on Operating Systems Principles*, New York (2003).
9) Brumley, D. and Song, D.: Privtrans: Automatically Partitioning Programs for Privilege Separation, *Proc. 13th USENIX Security Symposium*, San Diego (2004).
10) Chen, S., Xu, J., Sezer, E., Gauriar, P. and Iyer, R.: Non-Control-Data Attacks Are Realistic Threats, *Proc. 14th USENIX Security Symposium*, Baltimore (2005).
11) Chen, X., Garfinkel, T., Lewis, E.C., Subrahmanyam, P., Waldspurger, C.A., Boneh, D., Dwoskin, J. and Ports, D.R.K.: Overshadow: A Virtualization-Based Approach to Retrofitting Protection in Commodity Operating Systems, *Proc. 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, Seattle (2008).
12) England, P., Lampson, B., Manferdelli, J., Peinado, M. and Willman, B.: A Trusted Open Platform, *Computer*, Vol.36, No.7, pp.55–62 (2003).
13) Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M. and Boneh, D.: Terra: A Virtual Machine-Based Platform for Trusted Computing, *Proc. 19th ACM Symposium on Operating Systems Principles*, New York (2003).
14) Goldberg, I., Wagner, D., Thomas, R. and Brewer, E.: A Secure Environment for Untrusted Helper Applications, *Proc. 6th USENIX Security Symposium*, San Jose (1996).
15) Härtig, H., Hohmuth, M., Feske, N., Helmuth, C., Lackorzynski, A., Mehnert, F. and Peter, M.: The Nizza Secure-System Architecture, *Proc. 1st International Conference on Collaborative Computing*, San Jose (2005).
16) Jain, S., Shafique, F., Djeric, V. and Goel, A.: Application-Level Isolation and Recovery with Solitude, *Proc. 3rd European Conference on Computer Systems*, Glasgow (2008).
17) Jiang, X., Wang, X. and Xu, D.: Stealthy Malware Detection Through VMM-Based "Out-of-the-Box" Semantic View Reconstruction, *Proc. 14th ACM Conference on Computer and Communications Security*, Alexandria (2007).
18) Kim, G. and Spafford, E.: The Design and Implementation of Tripwire: A File System Integrity Checker, *Proc. 2nd ACM Conference on Computer and Communications Security*, Fairfax (1994).
19) Lie, D., Thekkath, C. and Horowitz, M.: Implementing an Untrusted Operating System on Trusted Hardware, *Proc. 19th ACM Symposium on Operating Systems Principles*, New York (2003).
20) Lie, D., Thekkath, C., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J. and Horowits, M.: Architectural Support for Copy and Tamper Resistant Software,

© 2009 Information Processing Society of Japan

Proc. 9th International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge (2000).
21) NSA: Security-Enhanced Linux. http://www.nsa.gov/selinux/
22) Parampalli, C., Sekar, R. and Johnson, R.: A Practical Mimicry Attack Against Powerful System-Call Monitors, *Proc. 2008 ACM Symposium on Information, Computer and Communications Security*, Tokyo (2008).
23) Patil, S., Kashyap, A., Sivathanu, G. and Zadok, E.: I^3FS: An In-Kernel Integrity Checker and Intrusion Detection File System, *Proc. 18th USENIX Conference on Security Administration*, Atlanta (2004).
24) Potter, S., Nieh, J. and Selsky, M.: Secure Isolation of Untrusted Legacy Applications, *Proc. 21st Large Installation System Administration Conference*, San Diego (2007).
25) Provos, N.: Improving Host Security with System Call Policies, *Proc. 12th USENIX Security Symposium*, Washington, DC (2003).
26) Rosenblum, N., Cooksey, G. and Miller, B.: Virtual Machine-Provided Context Sensitive Page Mappings, *Proc. 2008 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, Seattle (2008).
27) Saxena, P., Sekar, R. and Puranik, V.: Efficient Fine-Grained Binary Instrumentation with Applications to Taint-tracking, *Proc. 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, Boston (2008).
28) Ta-Min, R., Litty, L. and Lie, D.: Splitting Interfaces: Making Trust Between Applications and Operating Systems Configurable, *Proc. 7th USENIX Symposium on Operating Systems Design and Implementation*, Seattle (2006).
29) Wagner, D. and Dean, D.: Intrusion Detection via Static Analysis, *Proc. 2001 IEEE Symposium on Security and Privacy*, Oakland (2001).
30) Wagner, D. and Sato, P.: Mimicry Attacks on Host-Based Intrusion Detection Systems, *Proc. 5th ACM Conference on Computer and Communications Security*, Washington, DC (2002).
31) Xu, W., Bhatkar, S. and Sekar, R.: Taint-enhanced Policy Enforcement: A Practical Approach to Defeat a Wide Range of Attacks, *Proc. 15th USENIX Security Symposium*, Vancouver (2006).
32) Yang, J. and Shin, K.: Using Hypervisor to Provide Data Secrecy for User Applications on a Per-Page Basis, *Proc. 2008 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, Seattle (2008).
33) Zhao, X., Borders, K. and Prakash, A.: Towards Protecting Sensitive Files in a Compromised System, *Proc. 3rd IEEE International Security in Storage Workshop*, San Francisco (2005).
34) 滝澤祐二，光来健一，千葉　滋，柳澤佳里：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，*IPSJ Transactions on Advanced Computing Systems*, Vol.1, No.2, pp.275–285 (2008).
35) 尾上浩一，大山恵弘，米澤明憲：システムコール制御に基づく仮想マシン間サンドボックスシステム，*IPSJ Transactions on Advanced Computing Systems*, Vol.2, No.1, pp.33–52 (2009).

付　　録

A.1　セキュリティポリシ文法

PolicyFile	→	ShadowExecutable `shadowMemoryFile` : ShadowMemFile CalleeSpec∗
ShadowExecutable	→	`executable` : *pathOutsideVM*
ShadowMemFile	→	`all` ShadowFileSpec∗
		| `partially` PartSpec+ ShadowFileSpec∗
PartSpec	→	`kernelSeg` : KernSeg+ | `mmapRegion` | ElfSec | ElfSeg
		| `readOnlyRegions` : RORegion+
KernSeg	→	`text` | `data` | `brk` | `stack`(`DOWN`|`UP`, *size*) | `env` | `arg`
ElfSeg	→	`ELFSeg` : *segNum*+
ElfSec	→	`ELFSec` : *secName*+
RORegion	→	`text` | ElfSeg | ElfSec | `mmapRegion`
ShadowFileSpec	→	`shadowFile` : SFSpec+
SFSpec	→	FileSpec <Permission+>
FileSpec	→	`pathName`(*insideVM*, *outsideVM*)
		| `pathPrefix`(*insideVM*, *outsideVM*)
Permission	→	`read` | `write` | `create` | `append`
CalleeSpec	→	`calleeExecve` : ExecveSpec+
ExecveSpec	→	*pathInsideVM policyFile*

(平成 21 年 1 月 27 日受付)
(平成 21 年 6 月 3 日採録)

© 2009 Information Processing Society of Japan

尾上　浩一

1979 年生．2005 年東京大学大学院情報理工学系研究科コンピュータ科学専攻修士課程修了．現在，東京大学大学院情報理工学系研究科コンピュータ科学専攻博士課程．興味はオペレーティングシステムや仮想マシンモニタ等のシステムソフトウェア，セキュリティ．

大山　恵弘（正会員）

1973 年生．2001 年東京大学大学院理学系研究科情報科学専攻修了．博士（理学）．科学技術振興事業団研究員，東京大学大学院情報理工学系研究科助手を経て，現在，電気通信大学情報工学科准教授．興味はシステムソフトウェア，セキュリティ，プログラミング言語，並列分散処理．

米澤　明憲（正会員）

1947 年生．1970 年東京大学卒業．1977 年 MIT 計算機科学科博士課程修了．Ph.D. 1988 年東京大学理学部情報科学科教授に着任．日本ソフトウェア科学会理事長，フェロー，功労賞受賞，ドイツ国立情報科学技術研究所科学顧問等を歴任．現在，（独）産業技術総合研究所情報セキュリティ研究センター副センター長，東京大学情報基盤センター長を兼務．第 12 期日本学術会議会員．マイクロソフト本社 Trust-worthy Computing Academic Advisory Board メンバ．2008 年国際オブジェクト技術協会（AITO）Dahl-Nygaard 賞受賞．

© 2009 Information Processing Society of Japan