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. 1　33–52 (Mar. 2009)

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

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侵入検知システムのようなセキュリティシステムはアプリケーションの不正な振舞いやデータの改竄を検出・防止することができる．しかし，セキュリティシステム自体も攻撃対象となりうる．セキュリティシステムへの攻撃を防ぐためには仮想マシンモニタ（VMM）が有用である．VMM の導入により仮想マシン（VM）単位で実行環境を隔離できるようになり，制御対象となる VM の外側でセキュリティシステムを稼働させられる．しかし，VM の外側で取得できるイベントや実行状態は割込みやレジスタやメモリの値のようなハードウェアレベルのものに限られる．本論文では，VM の外側から制御対象とするプログラムが実行したシステムコールを制御し，VM 内の安全性を向上させるセキュリティシステム *ShadowVox* を提案する．ハードウェアレベルの実行状態から制御対象とするプログラムの実行状態を復元するために，制御対象となる OS におけるプロセスとシステムコールに関する情報を用いる．実行されたシステムコールは利用者によって作成されたセキュリティポリシによって制御される．我々は Xen を用いて ShadowVox を IA-32，AMD64 アーキテクチャに実装し，サーバプログラムやセキュリティシステムに適用した．

Inter-Virtual Machine Sandboxing System Based on System Call Interposition

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

Security systems such as sandboxing systems and intrusion detection systems can monitor and control behaviors of untrusted programs and prevent attackers from tampering with computing resources. However, security systems can also be attacked. To protect security systems, it is useful for security systems to cooperate with a virtual machine monitor (VMM). A VMM can isolate virtual machines (VMs) for security systems from VMs for untrusted programs. However events and execution states of untrusted programs observed outside of VMs are hardware-level events such as interrupts, and states such as register and memory values. In this paper, we propose *ShadowVox*, a security system that controls system call execution of untrusted programs from outside of untrusted VMs. To bridge the semantic gap between hardware-level states and OS-level states, we use information on untrusted OSes related to processes and system calls. System calls are intercepted by the VMM and controlled according to security policies described by users. We implemented ShadowVox using Xen on IA-32 and AMD64 architectures, and applied the system to server applications and security systems.

1. はじめに

悪意を持つ者による計算機システムへの不正侵入，コンピュータウィルス，ルートキット，ワームなどによる攻撃を検知・防止するためには，アプリケーションの動作を監視・制御するセキュリティシステムの利用が効果的である．これまで数多くのサンドボックスシステム[4),11),13),23),29)]，侵入検知/防止システム[5),9),35)]，アンチウィルスシステム[24),32),33)] などのセキュリティシステムが提案されている．

しかし，セキュリティシステムの普及とともに，攻撃者もセキュリティシステムの存在を意識するようになってきている[14),17),26)]．攻撃者は，不正プログラムの存在や振舞いをセキュリティシステムに検知されない手法やセキュリティシステム自体への攻撃手法を実現しようと試みる．また，セキュリティシステム自身もプログラムであるため，脆弱性を含みうる[1)–3)]．さらに，セキュリティシステムには管理者権限を必要とするものも存在する．この場合にはセキュリティシステムが乗っ取られると，システム全体が乗っ取られてしまうことになる．このため，セキュリティシステムの安全性を高めることは他のアプリケーションプログラムの安全性を高めることよりも重要である．

仮想マシンモニタ（VMM）の利用はセキュリティシステムの安全性を向上させるために有用な手段の 1 つである．VMM の 1 つの特長である仮想マシン（VM）単位の実行環境の隔離を利用すると，セキュリティシステムの制御対象であるプログラム（制御対象プログラム）の外側でセキュリティシステムを稼働させることができる．これにより，たとえ攻撃

†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

者が VM 内のプログラムを乗っ取っても，制御対象プログラムが稼働する VM（制御対象 VM）の外側で動作するセキュリティシステムの制御を乗っ取ることは困難になる．この強い隔離という特長を安全性向上のために利用したシステムがこれまでいくつか提案されている[10),22),31)]．

制御対象 VM の外側でセキュリティシステム（制御プログラム）を稼働させるうえでの課題は，VMM が捕捉可能な特権命令の実行や割込み，例外のようなハードウェアレベル（低レベル）のイベントから，OS レベル（高レベル）の制御対象プログラムの実行状態および振舞いを復元することである．セキュリティシステムの多くは，ハードウェアレベルのイベントではなく，プロセス，ファイル，ネットワーク資源などの OS レベルの資源に関する制御を行う．制御対象 VM の外側から制御対象プログラムの高レベルな実行状態および振舞いを識別することは容易ではない．

本論文では，制御対象 VM の外側で稼働するプログラムが制御対象 VM 内で発行されたシステムコールの実行を制御することにより，プログラムの振舞いを制御するセキュリティシステム ShadowVox を提案する．制御対象プログラムが発行したシステムコールは利用者が記述したセキュリティポリシに従って制御される．我々は VMM が取得可能な低レベルなイベントや実行状態から制御プログラムで利用する高レベルな実行状態を復元するために，制御対象 VM 内で稼働する OS（制御対象 OS）に関する情報を利用する．この情報には，制御対象 OS に依存した，レジスタやメモリからプロセスの実行状態を取得する方法，システムコールの呼び出し規約，プロセスやシステムコールに関連するデータ構造が含まれる．ShadowVox ではデータ構造の情報を制御対象 OS イメージのコンパイル時に自動生成する．また，VMM がシステムコールを捕捉するために制御対象 OS のカーネルコードのバイナリ書き換えを用いる．これにより，制御対象 OS のソースコードを修正することなく，システムコールの開始，終了時を VMM が捕捉できるようになる．セキュリティポリシにはどのシステムコールを制御対象とするか，捕捉したシステムコールをどのように制御するかを記述する．

我々は，VMM として Xen[7)] を利用し，ShadowVox の設計および実装を行った．ShadowVox は CPU アーキテクチャとして IA-32，AMD64 を，制御対象 OS として Linux をサポートしている．ShadowVox の評価では，既存のサーバプログラムやセキュリティシステムへ適用するとともに，ShadowVox がセキュリティシステム自体に対する攻撃から保護できることを確認する．さらに，マイクロベンチマーク，アプリケーションベンチマークを用いて，システムコール捕捉にともなうオーバヘッドを測定する．

以降，本論文は以下のように構成される．2 章で制御対象 VM の外側からの実行制御について述べる．3 章で提案システム ShadowVox について説明をした後，4 章で ShadowVox の実装について述べる．5 章で ShadowVox に関する安全性を議論し，6 章で ShadowVox を評価する．関連研究について 7 章で述べる．8 章でまとめと今後の課題について述べる．

2. 制御対象となる VM の外側からの実行制御による安全性の向上

2.1 VMM とセキュリティシステムの協調

VMM とセキュリティシステムを組み合わせて用いることは VM 内の実行環境の安全性を向上させるために効果的である．ここではセキュリティシステムと制御対象プログラムの位置関係によって 2 つの手法に分類し，各々の特徴を述べる．

1 つは制御対象 VM 内でセキュリティシステムを稼働させる手法である．この手法の利点は，セキュリティシステムが OS およびプログラムに関する OS レベルの情報を利用できることである．セキュリティシステムはプロセスの振舞いやファイルやネットワークなどの計算資源操作を OS レベルで制御することができる．ほかにも既存のセキュリティシステムを修正することなく利用できるという利点がある．しかし，攻撃者がセキュリティシステムを比較的容易に停止・無力化できてしまうという問題がある．

もう 1 つはセキュリティシステムを制御対象 VM の外側で稼働させる手法である．この手法では，VM 単位の強い隔離により，たとえ攻撃者が制御対象 OS の管理者権限を奪取したとしても，セキュリティシステムを停止・無力化させることが困難である．しかし，この手法ではセキュリティシステムが利用できる実行状態が，レジスタやメモリなどのハードウェアレベルの実行状態に限られるという問題がある．セキュリティシステムがハードウェアの実行状態をもとに OS レベルの振舞いを制御することは単純ではない．

2.2 提案システムが提供する安全性

本論文ではシステムコールの実行を制御するセキュリティシステム自体の安全性を向上させるシステムを提案する．セキュリティシステムを他のプログラムと同一 VM 内で稼働させた場合，同一 VM 内の管理者権限を有する制御対象外プログラムが乗っ取られたときに以下の攻撃の脅威が生じる．たとえば，攻撃者は乗っ取った制御対象外プログラムからセキュリティシステムを強制終了させた後，制御対象プログラムを攻撃することができる．また，セキュリティシステムが利用するポリシファイルやデータベースファイルを書き換え，セキュリティシステムを無効化することもできる．セキュリティシステムを制御対象 VM と異なる VM で稼働させることでこれらの攻撃を防ぐことができる．

しかし，ハードウェアレベルの実行状態から制御プログラムが必要とするプロセスやシステムコールに関する実行状態を正確に取得し，その振舞いを制御することは容易ではない．提案システムでは制御対象 VM の外側で稼働するセキュリティシステム（制御プログラム）がシステムコールの実行制御に基づき制御対象プログラムの安全性を向上させる．このために，制御プログラムはハードウェアレベルの実行状態から制御対象プログラムの実行状態を復元し，実行されたシステムコールを制御する必要がある．これを実現するために我々は制御対象 OS の情報を用いる．これにより，制御対象 VM 内のプロセス粒度の安全性を提供し，かつセキュリティシステム自体に対する攻撃も保護できる．

我々は制御対象 VM 内で稼働する既存のセキュリティシステムがアプリケーションの実行を制御し，セキュリティシステム（制御プログラム）が既存のセキュリティシステムの実行を制御するという運用も重視している．これにより，システムコールの実行制御による安全性に加え，アンチウィルスシステムや侵入検知システムなどのセキュリティシステムが提供する安全性が追加される．一方，管理者権限を必要とすることが多いそれらのセキュリティシステムの実行は制御対象 VM の外側で稼働する制御プログラムにより制御される．

3. 提案システム

3.1 基 本 構 成

提案システム ShadowVox の基本構成を**図 1** に示す．ShadowVox は VMM 内コンポーネント（SV-core）と制御 VM 内のシステムコール制御用プログラム群からなる．利用者が記述したセキュリティポリシに基づき，制御プログラムは制御対象プログラムが発行したシステムコールの実行を制御する．

システムコールの捕捉，制御対象プログラムの判別およびセキュリティポリシで各システムコールに対応づけられた処理（対応処理）は SV-core で行う．一方，セキュリティポリシの管理，発行されたシステムコールの制御方法の決定は制御 VM で行う．これらの操作を制御 VM 内で行うことにより，VMM が管理するデータを軽減させることができる．さらに，制御機構を柔軟に変更できる．

3.1.1 VMM 内コンポーネント SV-core

SV-core は制御 VM 内で実行されたシステムコールの開始・終了を捕捉する．終了時の捕捉は fork や accept などのシステムコール終了後に実行状態を検査するために必要となる．さらに，対応処理を実行するためにもシステムコール終了時の捕捉が必要となる．発行されたシステムコールを制御プログラムで制御する必要がある場合には制御 VM に通知する．

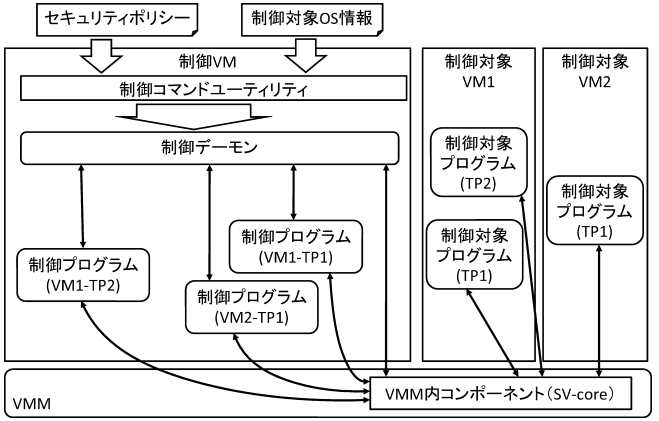

図 1　ShadowVox の構成
Fig. 1　ShadowVox structure.

また，制御対象プログラムが実行を開始したときにも制御 VM に通知する．SV-core はこれらの制御に必要なセキュリティポリシや制御対象 OS に関する情報を保持する．

3.1.2 制 御 VM

制御 VM 内では制御デーモン，制御プログラムを稼働させる．さらに実行制御のための制御コマンドユーティリティを提供する．

制御デーモンはセキュリティポリシや制御対象 OS 情報を管理する．さらに，制御プログラムの稼働や管理も行う．SV-core から受信した制御対象プログラムの制御開始通知ごとに制御プログラムを稼働させる．制御プログラムは SV-core と連携して制御対象プログラムのシステムコールの実行を制御する．

制御 VM の管理者は制御コマンドユーティリティを用いて，制御対象 OS やセキュリティポリシおよび制御対象プログラムのコマンド名を制御デーモンに通知する．制御コマンドユーティリティには次のものが用意されている．

- `vps`：このコマンドは，引数として制御対象 VM の ID を受け取り，制御対象 VM 内のプロセスの情報を出力する．プロセスリスト，各プロセスのプロセス ID，コマンド名，ユーザ ID などの情報が出力される．直感的には，このコマンドは制御対象 VM の外から UNIX の ps コマンドを実行できるようにしたものである．
- `vconf`：このコマンドは制御対象 OS に関する情報を ShadowVox に通知するためのコ

```
[dom0] # vconf targetOS.img
           targetOS_info.txt syscall_info.txt targetOS_symbol.txt
[dom0] # vcntl 1 targetOS.img
[dom0] # vstart 1 apache2 policy.txt log.txt
```

図 2　利用例
Fig. 2　Usage example.

マンドである．ShadowVox では OS カーネルのバイナリイメージごとに制御対象 OS に関する情報を管理する．引数には制御対象 OS のバイナリイメージと 3 つの実行制御に必要な情報を指定する．引数として指定されたバイナリイメージからハッシュ値を生成し，それを制御対象 OS の識別子として用いる．実行制御に必要な情報には制御対象 OS におけるプロセスとシステムコールに関する情報が含まれる．もう 1 つは捕捉するシステムコールの開始と終了位置に関する制御対象 OS のシンボル情報である．

- `vcntl`：このコマンドは指定した VM 内で実行されたシステムコールを制御するための設定を行うコマンドである．引数には制御対象 VM の ID と制御対象 OS のバイナリイメージを指定する．このコマンドが実行されると，バイナリイメージからハッシュ値が生成され，ShadowVox で管理している制御対象 OS のハッシュ値と比較される．これにより，ShadowVox が制御対象 OS に関する制御に必要な情報を識別できるようになる．さらに，この時点で制御対象 OS のバイナリイメージにおけるシステムコール開始と終了位置にある命令が書き換えられる．
- `vstart`：このコマンドは制御対象プログラムの実行制御を開始するためのコマンドである．引数には制御対象 VM の ID，制御対象プログラムのコマンド名，セキュリティポリシが記述されたファイル名を指定する．実行制御ログをとりたい場合にはログファイル名も指定することが可能である．

3.1.3　利　用　例

図 2 では，提案システムにおいて，制御対象 VM（VM ID: 1）内の制御対象プログラム `target_prog` の実行を制御する場合の流れが示されている．`vconf` で利用者は制御対象 OS に関する情報を制御デーモンに通知している．このコマンドには制御対象 OS のバイナリイメージ（`targetOS.img`）と制御対象 OS に関する 3 つの情報を与える．その 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．`vcntl` では制御対象 VM の ID と制御対象 OS のバイナリイメージ（`targetOS.img`）を与え，制御対象 VM と制御対象 OS の情報を関連付ける．この例では `vconf` で登録された制御対象 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．さらに，この時点で制御対象 OS のバイナリイメージも書き換えられ，制御対象 VM 内で実行されたシステムコールを捕捉できるようになる．`vstart` では制御対象プログラムである `apache2`，セキュリティポリシを記述したファイル（`policy.txt`）と捕捉したシステムコール情報を保存するログファイル（`log.txt`）を指定し，実行制御の開始要求を発行している．

3.2　制御に必要な OS 情報

ShadowVox は制御対象 OS に関する 2 つの知識を用いて実装されている．1 つはプロセス管理についての知識であり，SV-core が利用する．たとえば，レジスタやメモリの実行状態からプロセスの実行状態を取得に関する方法がこれに含まれる．もう 1 つはシステムコールに関する知識であり，SV-core と制御プログラムが利用する．たとえば，各システムコールの引数に関する知識がこれに含まれる．SV-core ではレジスタやスタックを用いてシステムコールの引数渡しや返り値の設定をする方法も必要となる．

ShadowVox では，アーキテクチャや制御対象 OS カーネルに依存する以下のデータを制御対象 OS の利用者から提供してもらう必要がある．

- プロセス関連のデータ構造：
  プロセスに関するデータ構造のメモリレイアウトやカーネルスタックサイズの情報がこれらに含まれる．たとえば，プロセス管理データ構造体におけるプロセス ID やユーザ ID の変数のオフセットとその大きさが必要である．SV-core では，制御対象 VM 内のプロセスの実行状態の取得や対応処理でこのデータ構造を利用する．このデータ構造は制御対象 OS カーネルのバージョンとコンパイルオプション，アーキテクチャに依存する．ShadowVox はこのデータ構造に関する情報を自動的に生成するためのプログラムを提供している．
- システムコールに関連した情報：
  この情報には制御対象 OS が提供するシステムコールの数，システムコール名と番号の関係，エラーコードが含まれる．スタック上に待避されるシステムコール引数のメモリレイアウトも含まれる．さらに，各システムコールが受け取るポインタ引数に関する情報もこれに含まれる．ポインタ引数はファイル名やソケット構造体を指す．ポインタ引数の情報には引数番号とポインタが指し示す領域の大きさが含まれる．SV-core はシステムコール情報の取得や対応処理でこれらの情報を利用する．制御プログラムがセキュ

リティポリシに基づきシステムコールを制御するためにシステムコール名と番号の関係を利用する．これらの情報は制御対象 OS カーネルのバージョンとアーキテクチャに依存する．ShadowVox はシステムコール引数のメモリレイアウトやポインタが指し示す領域の大きさに関する情報を自動的に生成するためのプログラムを提供している．システムコール名と番号の関係などは制御対象 OS カーネルの関連するヘッダファイルを用いる．

- システムコールの開始・終了に関連した仮想アドレス：
  これは制御対象 OS カーネルによるシステムコール実行を SV-core が捕捉するために必要な仮想アドレスに関する情報である．SV-core はこの仮想アドレス情報を用いて制御対象 OS カーネルのバイナリコードを書き換える．ShadowVox は制御対象 OS のコンパイル時に生成されるカーネルシンボル情報からこの仮想アドレスに関する情報を取得する．

プロセスに関連したデータ構造とシステムコールに関する情報を生成するプログラムでは，制御対象 OS においてそれらのデータ構造が定義されているファイルや OS カーネルの設定ファイルを必要とする．このプログラムは制御対象 OS イメージのコンパイル時にプロセスやシステムコールに関連したデータ構造のメモリレイアウトや大きさを自動生成し，ファイルに出力する．制御対象 OS イメージを生成する利用者はこれらのデータ構造に関する知識を必要としない．ShadowVox ではこの自動生成されるファイルに加え，システムコールが定義されたヘッダファイル，カーネルシンボル情報を含むファイルを利用者から提供してもらう必要がある．

ShadowVox はプロセスやシステムコールに関する知識およびそれらのデータ構造に依存する．このため，本論文では制御対象 OS として Linux を用いているが，我々はこれらの知識やデータ構造を取得できる UNIX 系 OS にも適用可能であると考えている．

3.3　セキュリティポリシ

3.3.1　文　　法

ShadowVox におけるセキュリティポリシの文法の一部（すべての文法を A.1 に記載）と記述例をそれぞれ**図 3**，**図 4** に示す．システムコール名と引数の情報を用いたパターンマッチによりシステムコールの実行を制御する．セキュリティポリシの記述はシステムコールに関する知識を必要とする．我々はポリシ文法の抽象度を上げ，利用者により直感的にすることよりも，システムコールレベルの粒度で柔軟なポリシを記述できることを優先している．

ポリシファイルの先頭にはマクロの定義（DefMacro）を記述する．それに続く DefAction

PolicyFile	→	DefMacro ∗ `default` :DefAction PolOption∗ ModuleSpec∗
DefMacro	→	`includeMacro`(*path*)
DefAction	→	Action | `skip`
Action	→	`allow` | `deny`(*errno*) | `killProc`(*signum*)
		| `policyChange`(*policyfile*[, *logfile*]) | `ask`
PolOption	→	`execByPtracingProc` :PtAction | `signalMask`(*signums*)
PtAction	→	`detachProc` | `createNewMonitor`
ModuleSpec	→	ModuleName `default:`DefAction SysCallSpec*
ModuleName	→	`processMod` | `fileMod` | `networkMod`
SysCallSpec	→	*syscallname* `default:`DefAction ControlExpr*
ControlExpr	→	Cond∗ Action
Cond	→	FileCond | NetworkCond | Cond `and` Cond | Cond `or` Cond
FileCond	→	`fileEq`(*argnum*, *path*) | `filePrefixEq`(*argnum*, *pathprefix*)
NetworkCond	→	`ipaddrEq`(*ipaddr*) | `netaddrEq`(*netaddr*) | `portEq`(*portnum*)
		| `unixsockEq`(*unixsock*) | `unixsockPrefixEq`(*unixsock*)

図 3　セキュリティポリシ文法（抜粋）
Fig. 3　Security policy grammar (extracted).

の部分では，捕捉したシステムコールがどのパターンにもあてはまらなかった場合の対応処理（Action）を記述する．その次の部分には，オプションの指示（PolOption）を記述し，続いてモジュールに対する指示（ModuleSpec）を記述する．モジュールの指示，マクロの定義やオプションの指示については後述する．

対応処理の `allow` は捕捉したシステムコールの実行を継続させるための指示である．`deny` は指定された *errno* をエラーコードに設定し，捕捉したシステムコールの実行を失敗させるための指示である．`killProc` は捕捉したシステムコールを実行したプロセスに *signum* の番号のシグナルを送信するための指示である．制御対象プロセスのセキュリティポリシを変更する対応処理を行うには `policyChange` の *policyfile* にセキュリティポリシが記述されたファイルのパス名を指定する．実行制御ログも変更する場合にはログファイルのパス名を *logfile* に指定する．`ask` は，捕捉したシステムコールの制御方法を制御 VM の利用者に問い合わせるための指示である．実行制御ログを取るように指定されている場合，捕捉したシステムコールとそれに対する対応処理がログファイルに記録される．`skip` は，SV-core が

```
includeMacro("asm-generic/errno-base.h")
default:  deny(EPERM)
fileMod default:  allow
  open
    default:  allow
    fileEq(1,"/etc/passwd") or filePrefixEq(1,"/etc/cron.d")
      deny(EACCES)
processMod default:  allow
  execve
    default:  killProc(SIGKILL)
    fileEq(1,"/usr/bin/wserver") policyChange("wserver.pol")
```

図 4　セキュリティポリシの記述例
Fig. 4　Sample security policy.

捕捉したシステムコールを制御プログラムに通知せず，システムコールの実行を継続するための指示である．wait，poll や gettimeofday のようなセキュリティに関係が薄いシステムコールの実行通知を省略することにより，オーバヘッドを軽減できる．

ShadowVox のセキュリティポリシでは，システムコールをプロセス（`processMod`）など 11 のグループ（モジュール）に分類している．各システムコールは必ずいずれか 1 つのモジュールに分類される．ModuleSpec の DefAction にはモジュールに属するシステムコールがどのパターンとも一致しなかった場合の対応処理を指定する．

各システムコールに対する指示（SysCallSpec）では，システムコール名を *sysCallName* に文字列で記述する．次に，捕捉したシステムコール引数がどのパターンにもあてはまらなかった場合の対応処理を DefAction で指定する．システムコールの引数のパターン（ControlExpr）には，ファイル名（`fileEq`）や IP アドレス（`ipaddrEq`）などを指定できる．

マクロの定義（DefMacro）では，マクロ定数が定義されたヘッダファイルのパス名（*path*）を指定する．マクロにより，エラーコードやシグナル番号などをそれぞれ，EPERM や SIGTERM などの文字列で指定することができる．

PolOption では ptrace を利用して他のプロセス制御を行う制御対象プロセスが execve を実行した場合（`execByPtracingProc`）や制御対象外のプログラムから送信されるシグナルを制御する場合（`signalMask`）などに関する記述ができる．ptrace を用いて他のプロセスを制御するプログラムでは，制御対象が実行時に決まることが多い．このため，execve 以降の実行を制御した場合，ポリシが execve で実行されるプログラムに依存してしまう．ShadowVox では（`execByPtracingProc`）を用いることによって，制御対象に依存しないポリシを記述できるようになる．`detachProc` が指定された場合には execve で実行されるプログラムを制御対象から除く．`createNewMonitor` が指定された場合には，execve で実行を開始するプログラムに対し，別の制御プログラムが生成される．そして，生成された制御プログラムが execve で開始されるプログラムの実行を制御する．`signalMask` では制御対象プログラムへの送信を禁止したいシグナルを *signums* に指定する．たとえば，`signalMask` に SIGKILL を指定することによって，制御対象外プログラムからの強制終了に関するシグナル送信を防ぐことができる．

図 4 のセキュリティポリシでは，ファイルモジュールとプロセスモジュールに属するシステムコールのうち，パターンと一致しないものの実行は許可される．その他のモジュールに属するシステムコールの実行は失敗し，エラーコード EPERM が返される．ファイル `/etc/passwd` とディレクトリ `/etc/cron.d` の下のファイルのオープンは失敗し，エラーコード EACCES が返される．他のファイルのオープンは許可される．execve が発行された場合には，第 1 引数が `/usr/bin/wserver` であるときに限り `/usr/bin/wserver` 用のポリシ `wserver.pol` に切り換えて実行制御を継続する．その他の場合は SIGKILL シグナルを送信する．

3.3.2　セキュリティポリシの生成

制御対象プログラムに最適なセキュリティポリシを記述することは容易ではない．このため，ShadowVox では Systrace [29] や TOMOYO Linux [15] のように，過去の実行で得られたログファイルを用いてセキュリティポリシの作成を支援する．利用者はまず制御対象プログラムが発行するシステムコールを調査するため，PolicyFile の `default` を `allow` に指定し，制御対象プログラムが実行するシステムコールの情報をログファイルに記録する．その後，利用者は生成されたログファイルをもとに，セキュリティポリシを記述する．

制御対象プログラムが実行しうるすべてのシステムコールのパターンを実行させることは困難である．よって，過去の実行で発行されていないパターンのシステムコールに対するポリシをどう記述するかが問題になる．ShadowVox では，セキュリティポリシにおける PolicyFile や ModuleSpec，SysCallSpec の `default` に `ask` を指定することで，システムコールへの対応処理を実行時に決定できるようにしている．実行制御終了後，`ask` で実行時に決定した対応処理をもとに利用者がポリシファイルを変更する．

現在の ShadowVox では Systrace のようなセキュリティポリシを自動生成する仕組みは提供していない．TOMOYO Linux で支援する対象はセキュア OS のためのポリシであり，ShadowVox とは支援するポリシの対象が異なる．

3.4 特　　長

ShadowVox には以下の特長がある．

攻撃者による制御 VM の攻撃が困難　攻撃者が制御対象 VM 内のプログラムを乗っ取っても，制御 VM を直接攻撃することはできない．

制御対象プログラム単位での実行制御が可能　制御対象プログラムが不正処理や異常動作をした場合，制御対象プログラムのみに対応処理が適用され，VM 内の他のプロセスの実行状態は保たれる．一方，制御単位が VM であるシステムでは，対応処理は VM の再起動など粗粒度のものになる．VM の再起動には，その時点で動作中の他の正常プログラムを強制終了されてしまうなどの欠点がある．

複数の VM 内の制御対象プログラムを統一的に制御可能　1 つの VMM 上において同一の制御対象プログラムが複数の VM 内で動作している場合，制御 VM はそのプログラムを実行している複数の制御対象プログラムを，共通のセキュリティポリシによって制御できる．たとえば，ウェブサーバ Apache に対するセキュリティポリシを与えられた制御 VM が，複数の VM 内で動作する Apache の動作を制御することができる．

制御 VM の管理者が各 VM の安全性向上を支援可能　各 VM の利用者が異なる仮想ホスティングの場面において，管理者が制御対象プログラムを制御することにより，各制御対象 VM の利用者がセキュリティシステムを導入・運用する手間を軽減させることができる可能性がある．セキュリティシステムの導入・運用には，OS に関する詳しい知識が必要であることが多い．ShadowVox により，OS に詳しくない利用者の VM を，管理者が外部から制御して安全性を高めることが可能になる．

4. 実　　装

VMM として準仮想化技術を利用した Xen 3.0.3，VMM 上で稼働する OS（ゲスト OS）として Linux 2.6.16.19 を利用して提案システムの実装を行った．提案システムは IA-32 と AMD64，ゲスト OS の SMP 環境に対応している．

4.1 VMM・制御プログラム間の通信機構

プロセスの実行状態や対応処理の実行のために，SV-core と制御プログラムは共有されたリングバッファによって通信する．ShadowVox ではこのリングバッファを Xen が提供している VMM と VM 間の共有メモリを用いて実装している．SV-core から制御プログラムへのシステムコール捕捉の通知は Xen のイベント通信機構（event channel）を用いる．一方，制御プログラムから SV-core への対応処理の通知は Xen が提供する共有メモリを利用する．

4.2 システムコールの実行制御

SV-core は，制御プログラムに通知する必要のないシステムコールについては独自の判断により，制御対象プログラムの実行を再開させる．fork や accept などの終了時に処理を必要とするシステムコール以外の終了時の捕捉では制御プログラムへの通知を行わない．制御対象プログラムではないプロセスによるシステムコールや `skip` が指示されたシステムコールでも通知は行われない．fork や exit_group など一部の特殊なシステムコールは，`skip` の指示の有無にかかわらず制御プログラムへ通知する．

execve が発行された際には，その引数のプログラムが制御対象であるかどうかを検査する．制御対象プログラムが実行された場合には，制御デーモンへ通知し，制御プログラムによる実行制御が開始される．その他の場合には制御対象 VM の実行を再開させる．

制御プログラムがシステムコールの検査中は制御対象 VM 全体を停止させるのではなく，制御対象プロセスのみ停止するようにしている．これにより制御対象 VM 内の他のプロセスへの影響を小さくすることができる．

4.3 プロセスの実行状態の取得

SV-core は制御対象プロセスの実行状態を取得する必要がある．制御対象 VM 内のプロセスリストを取得する場合，制御対象 VM が実行中であったときには VMM が制御対象 VM を停止する．そして制御 VM から制御対象 OS カーネルへアドレス空間を切り換え，プロセスリストを取得し，各プロセスの実行状態を保存する．最後に，制御対象 VM から制御 VM へアドレス空間を切り換え，制御 VM に取得したプロセスリストの情報を通知する．

Linux カーネルは主なプロセス情報をカーネル内の `task_struct` 構造体で保持している．`task_struct` 構造体からプロセス ID，ユーザ ID などに関する情報を取得することができる．プロセスリストを取得する場合には，`task_struct` 構造体の `tasks` 変数を利用する．

VMM は制御対象 VM 内のプロセス情報を得るために `task_struct` 構造体へのポインタを取得する．Linux 2.6 では `task_struct` 構造体へのポインタが `thread_info` 構造体の `task` 変数となっている．`thread_info` 構造体へのポインタの取得方法はアーキテクチャと OS カーネルのバージョンに依存する．`task_struct` 構造体の各メンバ変数のオフセットや大きさはアーキテクチャと OS カーネルのバージョン，コンパイルオプションに依存する．

IA-32 ではカーネルスタックポインタを用いて `thread_info` 構造体へのポインタを取得す

る．Linux 2.6.20 以降ではスタックポインタを利用することも可能ではあるが，コントロールレジスタ（GS，FS）からも取得可能である．AMD64 では GS レジスタから `thread_info` 構造体へのポインタを取得する．

準仮想化技術を利用した Xen は VMM のメモリ空間をすべての VM で共有しているため，システムコールが実行されて VMM に処理が切り換わったときに仮想アドレス空間を変更する必要がない．このため，VMM が制御対象 OS カーネルが管理する `task_struct` 構造体を直接参照することが可能である．Linux OS カーネルではプロセス情報のようなカーネルのメモリ空間で管理されているデータを参照する場合にはページフォルトは生じない．

4.4　システムコールの実行状態の取得

SV-core はシステムコールの呼び出し規約に従い，レジスタとカーネルスタックを用いてシステムコール番号と引数を取得する．システムコールの呼び出し規約はアーキテクチャに依存し，システムコール番号はアーキテクチャと OS カーネルのバージョンに依存する．システムコール引数にはファイルパス名のようなポインタが含まれる．このため，システムコール番号とポインタ引数の対応，ポインタが指すメモリ領域の大きさがユーザ空間データを取得するために必要となる．

制御プログラムはシステムコール番号とシステムコールの引数を解釈する必要がある．システムコール番号とシステムコール名の対応を解釈するためにヘッダファイルを利用する．

SV-core がユーザ空間のメモリ領域を参照した場合にはページフォルトが生じる可能性がある．これに対応するため，SV-core が実行中の制御対象 OS のページテーブルを調査し，該当するユーザ空間のメモリ領域を含むページがページテーブルに存在するか確認する．該当ページが存在しない場合には，制御対象 OS にページフォルト処理を実行させるように SV-core が制御対象 OS の振舞いを変更する．制御対象 OS がページフォルト処理を完了した後 SV-core に処理が戻る．その後，SV-core は該当するメモリ領域を参照して引数の値を取得する．

4.5　システムコールへの対応処理

SV-core は制御プログラムからの通知または ShadowVox 自身の判断に従い，対応処理を実行する．

システムコールを失敗させる処理（`deny`(*errno*)）では，制御対象プロセスが実行するシステムコールをシステムコール開始時に getpid に変更する．getpid の終了処理時にエラーコードを *errno* で指定された値に設定する．accept，recvfrom，recvmsg の終了時の対応処理では，close を実行させるように制御対象プロセスの振舞いを変更する．close の終了後，指定したエラーコードを返り値に設定する．

スレッドグループにシグナルを送信する処理（`killProc`(*signum*)）では，制御対象プロセスへ *signum* で指定したシグナルを送信するように，実行するシステムコールを kill に変更する．スレッドにシグナルを送信する処理（`killThread`(*signum*)）では，*signum* で指定したシグナルを送信するように実行するシステムコールを tgkill に変更する．

セキュリティポリシを変更する処理（`policyChange`(*policyfile*)）では，システムコール捕捉時に適用するセキュリティポリシを *policyfile* に変更する．

対応処理を利用者に問い合わせる処理（`ask`）では，捕捉したシステムコール情報が標準出力に出力される．制御プログラムの利用者は Action から制御対象プログラムへの対応処理を標準入力に入力する．

4.6　システムコール捕捉機構

4.6.1　Linux におけるシステムコール手続き

IA-32 では，Linux 2.6 以降，2 つの方法（ソフトウェア割込み（INT 0x80）と SYSENTER 命令）でシステムコール呼び出しをサポートしている．システムコール処理の終了時には，割込みの終了手続き（IRET 命令経由）または SYSEXIT 命令を利用してユーザレベルに移行する．AMD64 の 64 ビットモード（AMD64）では，SYSCALL/SYSRET 命令を用いた方法でシステムコールを実行する．

ソフトウェア割込み処理に必要な特権レベルが 0 に設定されている場合，VMM はソフトウェア割込みによるシステムコール呼び出しを捕捉できる．SYSENTER，SYSCALL 命令は必ず特権レベルが 0 に移行するため，VMM がこれらの命令によるシステムコール呼び出しを捕捉できる．SYSEXIT，SYSRET 命令は特権レベルが 0 以外で実行された場合には例外が発生するため，これらの命令によるシステムコール終了処理を捕捉できる．IRET 経由のシステムコールの終了処理の場合，ユーザレベルに移行するまでに VMM により必ず捕捉される命令がないため，すべてのシステムコールの終了処理を捕捉できない．

4.6.2　Xen におけるシステムコール手続き

準仮想化技術を利用した IA-32 用の Xen（Xen-IA32）では，ソフトウェア割込みによるシステムコール手続きでは，VMM を経由せず直接ゲスト OS カーネルのシステムコール呼び出し手続きへ遷移させる．これより Xen-IA32 では VMM がソフトウェア割込みを用いたシステムコール呼び出しを捕捉することができない．また，Xen-IA32 では SYSEN-

TER/SYSEXIT 命令を利用したシステムコール手続きはサポートしていない[⋆1]．準仮想化技術を利用した AMD64 用の Xen（Xen-AMD64）では，SYSCALL を利用してシステムコールを開始する．Xen-IA32，Xen-AMD64 ともにシステムコールの終了時には IRET 経由でユーザレベルへ移行する．

4.6.3　提案システムにおけるシステムコール捕捉

VMM が捕捉できないシステムコール呼び出しと終了処理に対応するために，我々は制御対象 OS のバイナリコードを書き換えるための仕組みを導入している．バイナリ書き換えを用いることで，ゲスト OS のソースコードを修正することなく，システムコールの捕捉が実現できる．

バイナリ書き換えでは，捕捉が必要なシステムコール手続きに関連したバイナリコードの先頭を特権命令である HLT 命令に書き換える．システムコールを捕捉するバイナリコードの仮想アドレスは制御対象 OS カーネルのコンパイル時に生成される `System.map` から取得する．書き換え元の命令とアドレスは SV-core で管理する．書き換えた制御対象 OS カーネルのコード領域を保護するために，我々は書き換え以降，それらのコード領域を含むページを書き込み不可に設定する．書き換えた HLT 命令が実行されると，SV-core によりシステムコールの実行制御が開始される．捕捉したシステムコールの対応処理の決定後，VMM で保持しておいた書き換え前の命令をエミュレートし，制御対象プログラムの実行を継続する．

ソフトウェア割込みを利用したシステムコール呼び出し，IRET 経由のシステムコールの終了処理を捕捉するためにバイナリ書き換えを利用する．Xen-IA32 では，SYSENTER 命令を利用したシステムコール手続きに対応するための仕組みを追加している．制御対象 VM 内で SYSENTER 命令が実行されたとき，VMM 内の SYSENTER 命令処理コードに移行する．制御対象プロセスが SYSENTER 命令を実行した場合には，実行制御処理を開始する．

Xen-AMD64 では，SYSCALL 命令を利用したシステムコール呼び出しの実行を制御するため，VMM の SYSCALL 命令処理コードに実行制御コードを追加している．

5.　議論：提案システムに関する安全性

ShadowVox において信頼すべき構成要素は，VMM と制御 VM である．このため，これらの構成要素の中に脆弱性があった場合，システム全体が乗っ取られる可能性がある．ただし，VMM や制御プログラムは制御対象 VM の外側で稼働しているため，制御対象 VM 内からこれらの脆弱性を攻撃することは容易ではない．また，信頼すべき構成要素を攻撃されにくくするために，制御 VM の管理者に協力してもらい，制御 VM 内で稼働させるプログラムを制限してもらったり，制御 VM 内でのネットワーク経由のアクセスを禁止してもらったりすることも有用である．

我々は制御対象プログラムとしてユーザレベルプログラムを対象とし，それらが実行するシステムコールを制御プログラムによって制御する．ShadowVox はシステムコール実行の捕捉のために書き換えたコード領域を含むページの不正改竄を防止し，制御対象 OS に関する情報を用いて制御対象プログラムの実行状態を制御対象 VM の外側から直接取得する．制御対象プログラムに関する安全性は記述されたセキュリティポリシに依存する．

ShadowVox は制御対象 OS カーネルが不正改竄され，制御対象 OS カーネルが管理する制御対象プログラムのデータ構造を直接不正に変更されたり制御対象プログラムを強制終了させたりする攻撃を防止できない．これらの不正操作を行うカーネルレベルルートキットを用いた攻撃への対応は今後の課題である．また，TOCTTOU 競合状態を利用した攻撃やシンボリックリンクを利用した攻撃にも対応する必要がある．現在のシステムを拡張し，計算資源を操作しているプロセス以外の制御対象 VM 内のプロセスを停止させることでこれらの攻撃に対応することができる．しかし，この方法では制御対象 VM の性能低下が大きくなると考えられ，より効率的な方法で対応する必要がある．

6.　実　　験

6.1　既存のプログラムへの適用

次の 6 つのプログラムの起動から終了までの実行で得られたログファイルを用いて，セキュリティポリシを作成して各プログラムの実行を制御した．

- サーバプログラム：
  ウェブサーバ thttpd，Apache，メールサーバ Sendmail を利用した．thttpd では静的コンテンツの参照，Apache では静的コンテンツと動的コンテンツの参照を行った．Sendmail ではメールの受信と送信を行った．共通のセキュリティポリシを用いて，複数の VM 上で動作する Apache の実行制御も行った．
- セキュリティシステム：
  アンチウィルスツール ClamAV，ホスト型侵入検知システム Tripwire，ネットワーク

⋆1 Supervisor mode kernel のゲスト OS カーネルのみサポートしている．

型侵入検知システム Snort，システムコール制御プログラム strace と Systrace を利用した．ClamAV には，ウィルスデータベースの管理プログラム（freshclam），ウィルス検査を実行するプログラム（clamscan，clamd/clamdscan）が含まれる．freshclam によるウィルスデータベースの初期化と更新，clamscan，clamd/clamdscan を用いたウィルス検査を行った．Tripwire ではファイルシステム情報のデータベースの初期化と更新，ファイルの改竄検査を行った．Snort では Apache が利用するポート番号へのアクセスログを記録した．strace と Systrace では，ps，ls，cp のコマンドラインプログラムの実行と静的コンテンツを提供する thttpd の振舞いに対する監視を行った．

- サーバプログラムとセキュリティシステムの連携：
  Sendmail と clamd，clamav-milter を連携させ，ウィルスを含む添付ファイルを送信したときにメールのウィルス検査を行った．1 つの制御対象 VM で Sendmail と clamav-milter を稼働させ，各々に対し制御対象プログラムを制御 VM で稼働させた．別の制御対象 VM で clamd を稼働させ，それに対する制御対象プログラムも稼働させた．

6.2　提案システムによるセキュリティシステムの保護の確認

ShadowVox によるセキュリティシステムに対する攻撃耐性を評価するため，制御対象 VM 内で稼働する管理者権限を有するプログラムが乗っ取られた場合でも制御対象プログラムを実行制御できることを確認する．制御対象 VM 内で，制御対象プログラムとして Apache を，管理者権限を有するプログラムとして FTP サーバプログラム ProFTPD [28] を稼働させ，以下の実験を行った．

まず，ShadowVox を用いず，セキュリティシステムを制御対象 VM 内で稼働させた場合を想定して実験を行った．Systrace [29] により実行を制御し，ポリシで指定されていないファイルを Apache が読み出せないようにした．このとき，ProFTPD 1.2.8 におけるバッファオーバフローの脆弱性（CAN-2003-0831）を利用し，我々は異なる計算機から ProFTPD を乗っ取り，制御対象 VM の管理者権限でシェルを起動するプログラムを実行した．その後，Apache の公開ディレクトリにパスワードファイルを複製した．さらに，Systrace を無効化させ，この複製したファイルを外部から参照できるように，Systrace のポリシファイルを不正改竄した．これにより，依然として Systrace により Apache の実行が制御されていたにもかかわらず，我々は異なる計算機からパスワードファイルの中身を参照できた．

次に，ShadowVox により Apache の実行を制御する場合を想定して実験を行った．ShadowVox には，Systrace を用いた上記の実験とほぼ同じポリシを与えた．ProFTPD を乗っ取って起動したシェルを通じて，Systrace の場合と同様に，パスワードファイルを Apache の公開ディレクトリに複製した．その後，ShadowVox による制御を無効化するために，ポリシファイルの改竄を試みた．しかし，ポリシファイルは制御対象 VM の外側で管理されているため，ポリシファイルを不正に改竄することはできなかった．このため，ShadowVox による Apache の実行制御は無効化されることはなく，複製したパスワードファイルの中身を異なる計算機から参照することはできなかった．

6.3　システムコール制御にともなうオーバヘッドの計測

6.3.1　設　　定

ShadowVox により生じるオーバヘッドをマイクロベンチマークとアプリケーションベンチマークを用いて計測した．IA-32 の実験環境は CPU Pentium 4 3.0 GHz（Hyper-Threading 有効），1 GB メモリ，1 Gbps ネットワークカードである．AMD64 の実験環境は CPU Dual-Core AMD Opteron 2.8 GHz が 2 つ，8 GB メモリ，1 Gbps ネットワークカードである．

SV-core が制御通知を送信したとき，制御 VM が VMM 上で実行されていない可能性がある．制御 VM がつねに実行されるようにするため，制御 VM と制御対象 VM を実行する物理 CPU を分離するように設定した．本実験では VMM 上で制御 VM と 1 つの制御対象 VM を稼働させた．IA-32 では制御 VM と制御対象 VM は各々1 つの仮想 CPU を割り当て，メモリサイズは各々512 MB，256 MB に設定した．ADM64 では制御 VM と制御対象 VM は各々2 つの仮想 CPU を割り当て，メモリサイズはどちらも 512 MB に設定した．本実験では実行制御ログはとらないように設定した．

6.3.2　マイクロベンチマーク

我々は以下の各設定に対してシステムコール実行に要する時間の測定した．マイクロベンチマークの実験では，制御 VM で稼働する制御プログラムへの捕捉を通知する場合（Action が `allow`）と行わない場合（Action が `skip`）に対する実行時間を計測した．

- ShadowVox（SYSENTER）：IA-32 で SYSENTER 命令を利用．
- ShadowVox（INT0x80）：IA-32 でソフトウェア割込みを利用．
- ShadowVox（SYSCALL）：AMD64 で SYSCALL 命令を利用．
- Xen：IA-32，AMD64 で，Xen 上で Linux を稼働．
- Xen+ptrace：IA-32，AMD64 で，Xen 上で Linux を稼働させ，ptrace を用いた実験用のシステムコール制御プログラムを稼働．プロセスの生成・終了時にはプロセス管理用のハッシュテーブルへの追加・削除操作を実行．
- Linux：IA-32，AMD64 で Linux を稼働．
- Linux+ptrace：IA-32，AMD64 で，Linux を稼働させ，ptrace を用いた実験用のシ

表 1　マイクロベンチマークの結果（IA-32）（[マイクロ秒]）
Table 1　Microbenchmark results (IA-32) ([μsec]).

	ShadowVox (`allow`)		ShadowVox (`skip`)			Xen		Linux
	SYSENTER	INT 0x80	SYSENTER	INT 0x80	Xen	+ ptrace	Linux	+ ptrace
getpid	12.9	16.5	1.17	2.00	0.37	18.5	0.16	15.0
open	33.8	39.4	3.70	5.46	2.05	48.2	2.66	51.1
socket	35.7	43.4	4.98	6.45	3.01	46.1	3.48	44.2
fork	217	218	146	147	138	190	39.9	93.5

表 2　マイクロベンチマークの結果（AMD64）（[マイクロ秒]）
Table 2　Microbenchmark results (AMD64) ([μsec]).

	ShadowVox (`allow`) SYSCALL	ShadowVox (`skip`) SYSCALL	Xen	Xen + ptrace	Linux	Linux + ptrace
getpid	10.7	2.12	0.52	16.5	0.06	5.06
open	28.5	7.29	3.63	46.5	1.58	16.0
socket	28.9	8.49	4.99	36.8	2.06	12.2
fork	220	189	169	230	32.9	55.6

ステムコール制御プログラムを稼働.

以下の 4 つのマイクロベンチマークプログラムを各々10,000 回繰り返した.

- 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.
- open，close（open）：ホームディレクトリに置いた test.txt ファイル操作の捕捉に要する時間を計測する．SV-core がページフォールトの発生有無を検査し，ファイル名を取得する.
- socket，close（socket）：AF_INET，SOCK_STREAM のソケットの生成に要する時間を計測する．IA-32 の場合，socket の引数を制御対象プログラムのメモリ領域から取得する必要がある.
- fork，waitpid，exit_group（fork）：制御対象プログラムによるプロセス生成に関連した処理に要する時間を計測する．親プロセスが子プロセスを生成し，子プロセスの終了を待つ．SV-core と制御プログラムが新たに生成された子プロセスを制御対象に追加する．`skip` の場合には生成された子プロセスを制御対象に追加しない.

IA-32，AMD64 それぞれに対する実験結果を**表 1**，**表 2** に示す．表中のデータは一連のマイクロベンチマークプログラムの 1 回の実行に要する時間を表し，小さい方がシステムとしては有用である．表中の Xen と制御プログラムに捕捉を通知した場合（`allow`）の比較，Xen と制御プログラムに捕捉を通知しない場合（`skip`）の比較から，実行速度に与える影響はシステムコールの捕捉よりも VM 間でのイベント通知の方が大きいことが分かった．ptrace を用いた場合に比べ，ShadowVox による実行制御にともなう性能低下が小さい要因の 1 つは制御プログラムへの通知回数が ShadowVox の方が少ないことがあげられる．たとえば getpid の場合，ptrace は 1 回のシステムコール実行で getpid の実行開始時と終了時に 2 回の通知が発生する．一方，ShadowVox は getpid の実行開始時の 1 回のみ通知が発生する．SYSENTER，SYSCALL におけるシステムコールの開始時には例外は発生せず，命令をエミュレートする必要がない.

6.3.3　アプリケーションベンチマーク

既存のサーバプログラムとセキュリティシステムに対するセキュリティポリシを記述し，システムコールの実行制御にともなうオーバヘッドを計測した．記述したセキュリティポリシでは，Systrace[29] で自動生成されるセキュリティポリシで許可されるシステムコールに対し制御対象プログラムにより実行を制御するように指定した．ただし，自動生成されるポリシ記述を一部変更し，access，stat，lstat に対する実行制御は制御対象プログラムに通知しない（`skip`）ように記述した．この実験で用いたセキュリティポリシは実用規模であると我々は考えている．たとえば，Apache に対するポリシ（CGI に対するポリシを含まない）のルール数は，IA-32 で 134，AMD64 で 143 であった．また，ClamAV（clamd+clamdscan）に対するポリシのルール数は，IA-32 で 157，AMD64 で 142 であった.

ウェブサーバ

ApacheBench 2.0.40 を用いて制御対象 VM で動作する Apache 2.0.54 のウェブサービススループットを計測した．ApacheBench を実行した物理計算機は，Pentium 4 3.2 GHz（Hyper-Threading 有効），2 GB メモリ，100 Mbps ネットワークカードであり，制御対象 VM が稼働する計算機とは同一 LAN に設置した.

ApacheBench から静的コンテンツ（ファイルサイズ：1 KB，100 KB）と動的コンテン

ツ（CGI）の要求を行った．CGI ではリクエストを送信した計算機情報を取得し，それを整形して返した．要求数は 1 から 128 まで 2 の累乗で増加させた．Apache が CGI の実行を開始したときにセキュリティポリシを変更するように記述した（Apache に対するセキュリティポリシの一部を A.2 に記載）．

比較のために，IA-32 上では制御対象 VM 内で稼働する Apache を Systrace で制御した場合のスループットも計測した．Systrace は AMD64 には対応していない．本実験で利用した Systrace は ptrace を用いてシステムコールの実行を制御する．セキュリティポリシとしては Systrace で自動生成したものを用いた．ただし，ShadowVox に適用したセキュリティポリシで `skip` と指定されているシステムコールは，無条件で許可するような修正を加えた．

IA-32 上での実験結果を**図 5**，**図 6**，**図 7** に，AMD64 上での実験結果を**図 8**，**図 9**，**図 10** に示す．要求するファイルサイズが大きくなると実行制御にともなう性能低下は減少した．本実験では，ShadowVox は Systrace よりも小さいオーバヘッドでシステムコールの実行を制御することができた．Systrace では，修正された Linux カーネルを用いてシステムコール捕捉にともなうオーバヘッドを削減できる．しかし，準仮想化技術を用いた VMM を利用する場合，ゲスト OS となる Linux OS カーネルがすでに修正されており，Systrace が提供するカーネルパッチをそのまま適用することはできない．一方，ShadowVox はゲスト OS となる Linux OS カーネルを修正する必要がない．

ファイル検査を実行するセキュリティシステム

ShadowVox によりメールの添付ファイルのウィルス検査を実行する ClamAV とファイルの不正改竄を検査する Tripwire の実行を制御し，その際に生じるオーバヘッドを計測した．

まず，ClamAV 0.93 が提供している clamscan と clamd/clamdscan を用いたウィルス検査の実行時間を計測した．clamscan はウィルスデータベースを検査時に読み込み，ファイルのウィルス検査を実行するコマンドである．一方，clamdscan はウィルス検査要求を検査デーモン clamd に送信するコマンドである．clamd はウィルスデータベースを起動時にあらかじめ読み込んでおき，clamdscan からのウィルス検査要求の処理を行う．検査対象のファイルとして，ClamAV が提供しているテスト用の 5 つのウィルスファイルと 1 つの正常ファイルを用いた．clamscan と clamd/clamdscan 各々に対し，制御プログラムによる実行制御を行った．

さらに，Tripwire 2.4.1.2 を用いて制御対象 VM のファイルシステムの改竄検査の実行時間を計測した．IA-32 では検査対象として 28,169 のファイル（変更されたファイルは 21）を検査した．AMD64 では検査対象として 17,897 のファイル（変更されたファイルは 101）を検査した．

表 3　IA-32 上でのファイル検査時間（ClamAV，Tripwire）
Table 3　File checking time on IA-32 (ClamAV, Tripwire).

	clamscan [秒]	clamd + clamdscan [ミリ秒]	tripwire [秒]
Xen	3.79	4.72	60.0
ShadowVox	3.85	6.87	64.5

表 4　AMD64 上でのファイル検査時間（ClamAV，Tripwire）
Table 4　File checking time on AMD64 (ClamAV, Tripwire).

	clamscan [秒]	clamd + clamdscan [ミリ秒]	tripwire [秒]
Xen	1.88	2.71	24.7
ShadowVox	1.96	3.97	25.9

IA-32，AMD64 に対する実験結果を各々**表 3**，**表 4** に示す．実行時間が短い場合（clamd+clamdscan）では約 46%，その他の場合には約 8%以下にオーバヘッドを抑えられた．

ウェブサーバの実行を制御するセキュリティシステム

セキュリティシステムによりウェブサーバの実行を制御し，そのセキュリティシステムを ShadowVox により制御する実験を行った．制御対象 VM 内で thttpd 2.23 を稼働させ，6.3.3 項で用いた同一 LAN 内の物理計算機の ApacheBench から 1 KB のファイル要求を行った．要求数は 1 から 128 まで 2 の累乗で増加させた．

まず，セキュリティシステムとして Snort 2.3.2 を用いて，thttpd が利用するポート番号へのアクセスログを記録した．ShadowVox が Snort の実行を制御した場合におけるスループットを計測した．さらに ShadowVox が Snort と thttpd の両方の実行を制御した場合のスループットも計測した．

さらに，IA-32 では Systrace 1.6 を用いて thttpd に対するセキュリティポリシを自動生成し，thttpd の実行を制御し，ShadowVox が Systrace の実行を制御した場合におけるスループットを計測した．本実験では，thttpd の実行開始時に制御対象から thttpd を除く（`detachProc`）ようにセキュリティポリシで指定した．

Snort に関する実験結果を**図 11**，**図 12** に，Systrace に対する実験結果を**図 13** に示す．IA-32 では Snort，Snort+thttpd の実行制御に各々48%以下，57%以下のオーバヘッドが生じた．Systrace では 65%以下のオーバヘッドが生じた．AMD64 では各々21%，32%以

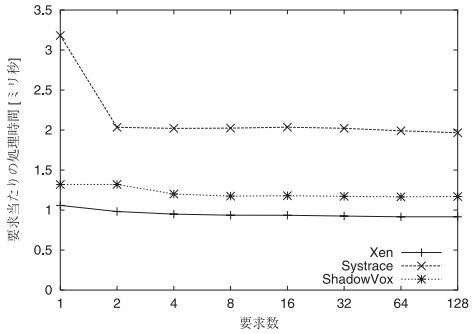

図 5　IA-32 上でのウェブサービススループット（1 KB）
Fig. 5　Web service throughput on IA-32 (1 KB).

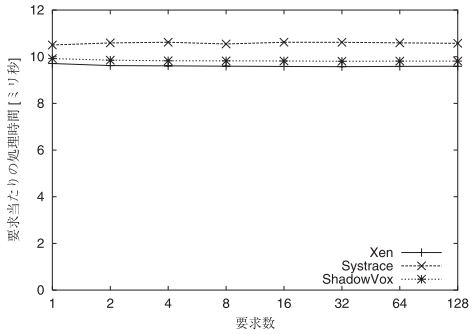

図 6　IA-32 上でのウェブサービススループット（100 KB）
Fig. 6　Web service throughput on IA-32 (100 KB).

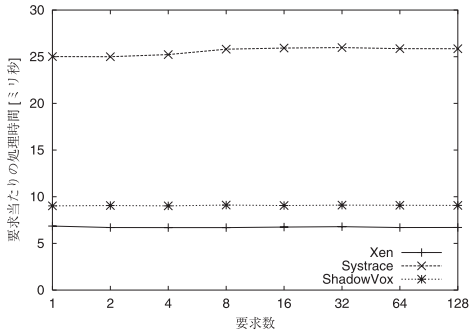

図 7　IA-32 上でのウェブサービススループット（CGI）
Fig. 7　Web service throughput on IA-32 (CGI).

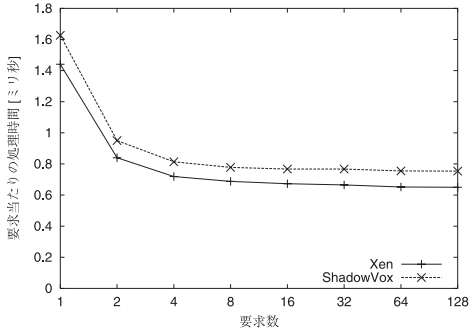

図 8　AMD64 上でのウェブサービススループット（1 KB）
Fig. 8　Web service throughput on AMD64 (1 KB).

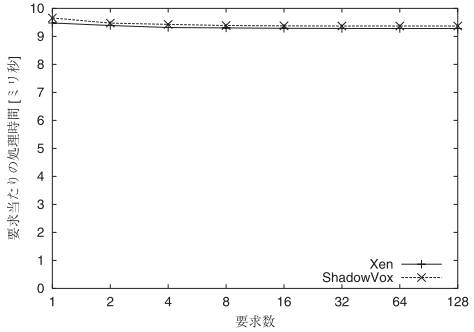

図 9　AMD64 上でのウェブサービススループット（100 KB）
Fig. 9　Web service throughput on AMD64 (100 KB).

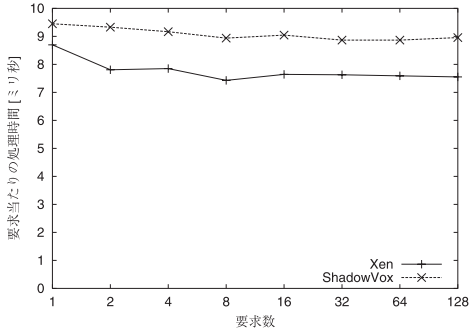

図 10　AMD64 上でのウェブサービススループット（CGI）
Fig. 10　Web service throughput on AMD64 (CGI).

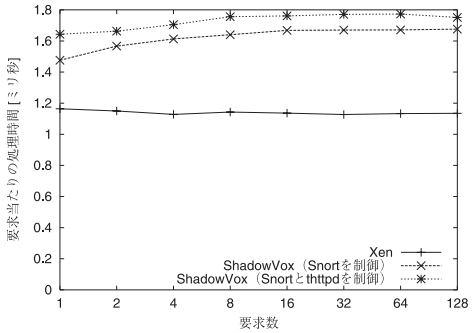

図 11　IA-32 上でのウェブサービススループット（Snort）
Fig. 11　Web service throughput on IA-32 (Snort).

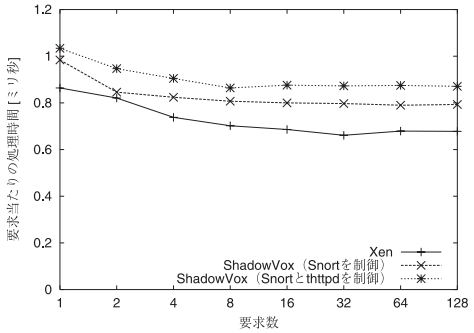

図 12　AMD64 上でのウェブサービススループット（Snort）
Fig. 12　Web service throughput on AMD64 (Snort).

下のオーバヘッドが生じた．

アプリケーションベンチマークの実験を通し，ShadowVox はシステムコール制御にともなうオーバヘッドはある程度生じるが，制御プログラムと制御対象プログラムの安全性を向上させることができると考えている．

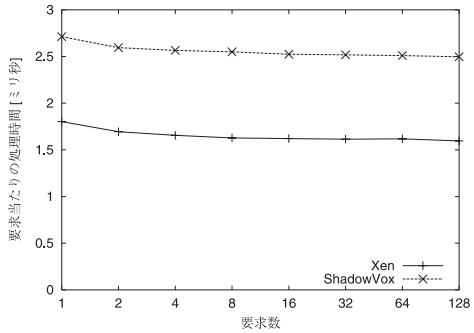

図 13　IA-32 上でのウェブサービススループット（Systrace）
Fig. 13　Web service throughput on IA-32 (Systrace).

7. 関 連 研 究

7.1　VM 単位の隔離による安全性の向上

sHype[31] は Chinese Wall や Type Enforcement のようなセキュリティポリシを用いて VM 間の情報流制御を行うシステムである．VMware ACE[34] は，VM のバイナリイメージとセキュリティポリシを組にしたパッケージの作成，配布により，VM の実行制御を一元管理するシステムである．NetTop[16] は SELinux[25] により提供される強制アクセス制御や情報流制御を用いることにより，VM 間を強く隔離することができる．Terra[10] はデータの機密性や完全性を高く保ちたいアプリケーションを専用の VM で稼働させ，他の VM からの不正操作を防ぐ．これらのシステムでは制御単位が VM であるのに対し，ShadowVox は VM 内のプロセス単位でより細かい制御を行うことができる．

7.2　VM の外側からの実行制御による安全性の向上

Livewire[12] は制御対象 OS の情報を用いて，制御対象 VM とは異なる VM 内でセキュリティシステムを稼働させる．Livewire では周期的な改竄検知，メモリや NIC へのアクセスのようなイベントを制御する．Livewire ではシステムコールの終了時のような VMM が捕捉できないイベントの安全性を検査することができない．ShadowVox ではバイナリ書き換えの仕組みを導入することで，制御プログラムが制御したいイベントを VMM が捕捉できるようにしている．さらに，Livewire が提供する VMM が実行する対応処理は VM の再起動や停止などの VM 単位の操作であるのに対し，ShadowVox はより粒度の細かいプロセ

ス単位の対応処理を提供する．

IntroVirt [21] は，VMM が既知の脆弱性を含むバイナリコード部分を捕捉できるように書き換え，利用者により作成されたその脆弱性に対する制御コード（Predicate）を実行する．捕捉時に実行する Predicate では制御対象 OS 情報を用いて制御対象プログラムの実行状態を操作する．ShadowVox は制御対象プログラムのシステムコール捕捉に基づいて実行を制御するため，未知の脆弱性を含む制御対象プログラムにも対処できる．

文献 6) は制御対象 OS カーネルが実行したソケット操作やファイル操作を監視し，VM 内で動作するプロセスへの攻撃に関する情報を収集する honeypot を提案している．このシステムでは監視処理がすべて VMM 内で行われるが，ShadowVox ではセキュリティポリシによる制御を制御 VM で行う．このため，セキュリティポリシを変更した場合にシステム全体を再起動させる必要がない．

VMwatcher [18] は制御対象 VM の外から制御対象 OS の情報を利用してマルウェアを検知するシステムである．VMM から参照可能なメモリやディスクブロックの実行状態からプロセスやファイルシステムの実行状態を構築する．構築した実行状態を既存のアンチマルウェアシステムに与えることにより改竄検知を行う．VMwatcher と異なり，ShadowVox はセキュリティポリシに基づいて制御対象プログラムの実行を制御する．また，VMwatcher は周期的に実行状態を検査するシステムであるのに対し，ShadowVox は制御対象プログラムのシステムコール実行に同期して実行を制御するシステムである．

Lycosid [20] は VMM 層から復元したプロセス情報と制御対象 OS 内から取得したプロセス情報の違いを利用して隠蔽されたプロセスの検出および識別を行うシステムである．Lycosid は Antfarm [19] で提案されている，VMM が捕捉可能なアドレス空間の切り換えに基づいて VMM から制御対象 OS 内のプロセスの生成，終了などの挙動を推測する．Lycosid は OS カーネルが管理するデータ構造に関する情報を利用しないため，制御対象 OS への依存度を小さくできるが，プロセス，ファイル，ネットワークなどの OS レベルの計算資源に対する処理を制御することは困難である．

文献 30) は，制御対象 VM の外側の制御 VM（仮想アプライアンス）でセキュリティシステムを稼働させる仕組みが提案されている．仮想アプライアンス内のセキュリティシステムが制御対象 VM によるネットワークアクセスを制御する．ShadowVox はネットワーク操作だけでなく，多様なシステムコールの実行を制御することができる．

XenAccess [27] は制御対象 VM の外側からメモリやディスク I/O 操作のような OS カーネルレベルの振舞いを監視するためのライブラリを提供する．他方，ShadowVox はシステムコール捕捉に基づいたより高レベルなプロセスレベルの振舞いを制御する．

7.3　システムコール捕捉に基づくセキュリティシステム

これまで，システムコール捕捉に基づき，プロセス，ファイル・ネットワークなどの計算機資源に関する操作の監視や制御を行う，侵入検知システム[8),35)] やサンドボックスシステム[13),29)] などのセキュリティシステムがさかんに提案・開発されている．これまでの既存研究で得られた先進的な知見は，ShadowVox におけるセキュリティポリシの改善に活かせると考えている．ShadowVox は，Janus [13] や Systrace [29] と同様，システムコールの捕捉機構とセキュリティポリシに基づく制御機構を分離したシステムである．一方，同一 OS 内の制御プログラムによりプロセスを制御するそれらのシステムとは異なり，VMM がシステムコールを捕捉し，制御 VM 内の制御プログラムが制御方法を決定する．このため，ShadowVox では，たとえ攻撃者が制御対象プログラムを乗っ取ったとしても，制御 VM 内の制御プログラムを直接攻撃することはできない．

8. まとめと今後の課題

本論文では，制御対象 VM の外側から VM 内で実行されたシステムコールを制御するサンドボックスシステム ShadowVox を提案した．そのシステムは VM の外側から VM 内のプログラムを制御するために，制御対象 OS におけるプロセスの管理方法やデータ構造，システムコールの呼び出し規約や引数などの情報を利用する．制御対象 OS のソースコードを修正することなくシステムコールを捕捉するために，制御対象 OS のカーネルコードのバイナリ書き換えを用いた．制御対象プログラムによって実行されたシステムコールは，利用者が与えるセキュリティポリシに従って実行制御が行われる．VMM として Xen を，制御対象 OS として Linux を用いて IA-32，AMD64 上で ShadowVox を実装し，サーバプログラムやセキュリティシステムに適用した．管理者権限で稼働する ProFTPD が乗っ取られた場合でも，同一の VM 内で稼働する Apache の実行は ShadowVox により依然として制御されていることを確認した．ShadowVox により実行を制御した Apache に対し，ApacheBench によって 1 KB のファイル要求を行った場合のオーバヘッドは IA-32 では最大 28%，AMD64 では最大 16%であった．

今後の課題としては，ファイル名に関するシステムコール引数の検査の精度の向上や TOCTTOU レースコンディションに関連した攻撃へ対応することがあげられる．ShadowVox における引数で渡されたファイルのパス名の絶対パスへの変換やシンボリックリンクの展開などを効率良く行う手法を検討する必要がある．また，セキュリティポリシを洗練さ

せることもあげられる．たとえば，制御対象プログラムの設定が各 VM で異なる場合でも共通のセキュリティポリシで制御できるようにするため，各 VM 固有の記述を行えるようにすることを検討している．さらに，カーネルレベルルートキットのようなカーネル領域のデータを改竄する攻撃への対応も今後の課題である．これに対しては ShadowVox で用いたバイナリ書き換えの適用を検討している．

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) Sophos AntiVirus Products Remote Heap Overflow vulnerability (2005). http://www.frsirt.com/english/advisories/2005/1244/
2) Snort GRE Packet Decoding Integer Underflow Vulnerability (CVE-2007-0251) (2007). http://cve.mitre.org/cgi-bin/cvename.cgi?name=2007-0251/
3) Multiple ClamAV Vulnerabilities (2008). http://www.secunia.com/advisories/29000/
4) Acharya, A. and Raje, M.: MAPbox: Using Parameterized Behavior Classes to Confine Untrusted Applications, *Proc. 9th USENIX Security Symposium*, Denver (2000).
5) Anagnostakis, K., Sidiroglou, S., Akritidis, P., Xinidis, K., Markatos, E. and Keromytis, A.: Detecting Targeted Attacks Using Shadow Honeypots, *Proc. 14th USENIX Security Symposium*, Baltimore (2005).
6) Asrigo, K., Litty, L. and Lie, D.: Using VMM-Based Sensors to Monitor Honeypots, *Proc. 2nd International Conference on Virtual Execution Environments*, Ottawa (2006).
7) 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).
8) Forrest, S., Hofmeyr, S.A., Somayaji, A. and Longstaff, T.A.: A Sense of Self for Unix Processes, *Proc. 1996 IEEE Symposium on Security and Privacy*, Oakland (1996).
9) Gao, D., Reiter, M. and Song, D.: Gray-box Extraction of Execution Graphs for Anomaly Detection, *Proc. 11th ACM Conference on Computer and Communications Security*, Washington, DC (2004).
10) 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).
11) Garfinkel, T., Pfaff, B. and Rosenblum, M.: Ostia: A Delegating Architecture for Secure System Call Interposition, *Proc. 11th Annual Network and Distributed System System Security Symposium*, San Diego (2004).
12) Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proc. 10th Annual Network and Distributed Systems Security Symposium*, San Diego (2003).
13) Goldberg, I., Wagner, D., Thomas, R. and Brewer, E.: A Secure Environment for Untrusted Helper Applications, *Proc. 6th USENIX Security Symposium*, San Jose (1996).
14) Halderman, J., Schoen, S., Heninger, N., Clarkson, W., Paul, W., Calandrino, J., Feldman, A., Appelbaum, J. and Felten, E.: Lest We Remember: Cold Boot Attacks on Encryption Keys, *Proc. 17th USENIX Security Symposium*, San Jose (2008).
15) Harada, T., Horie, T. and Tanaka, K.: Access policy generation system based on process execution history, Network Security Forum 2003 (2003). http://sourceforge.jp/projects/tomoyo/docs/nsf2003-en.pdf
16) Hewlett-Packard: NetTop (2004). http://www.hp.com/hpinfo/newsroom/press_kits/2004/security/ps_nettopbrochure.pdf
17) IBM Internet Security Systems: Cyber Attacks On The Rise: IBM 2007 Midyer Report, Technical report, IBM Internet Security Systems (2007).
18) 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).
19) Jones, S., Arpaci-Dusseau, A. and Arpaci-Dusseau, R.: Antfarm: Tracking Processes in a Virtual Machine Environment, *Proc. 2006 USENIX Annual Technical Conference*, Boston (2006).
20) Jones, S., Arpaci-Dusseau, A. and Arpaci-Dusseau, R.: VMM-based Hidden Process Detection and Identification using Lycosid, *Proc. 4th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, Seattle (2008).
21) Joshi, A., King, S., Dunlap, G. and Chen, P.: Detecting Past and Present Intrusions through Vulnerability-Specific Predicates, *Proc. 20th ACM Symposium on Operating Systems Principles*, Brighton (2005).
22) Kourai, K. and Chiba, S.: HyperSpector: Virtual Distributed Monitoring Environments for Secure Intrusion Detection, *Proc. 1st ACM/USENIX International Conference on Virtual Execution Environments*, Chicago (2005).
23) Liang, Z., Venkatakrishnan, V. and Sekar, R.: Isolated Program Execution: An Application Transparent Approach for Executing Untrusted Programs, *Proc. 19th Annual Computer Security Applications Conference*, Las Vegas (2003).
24) McAfee: McAfee — Antivirus Software and Intrusion Prevention Solutions.

http://www.mcafee.com/us/
25) NSA: Security-Enhanced Linux. http://www.nsa.gov/selinux/
26) 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).
27) Payne, B., Carbone, M. and Lee, W.: Secure and Flexible Monitoring of Virtual Machines, *Proc. 23rd Annual Computer Security Applications Conference Symposium on Information, Computer and Communications Security*, Florida (2007).
28) ProFTPD: ProFTPD — Highly configurable GPL-licensed FTP server software. http://www.proftpd.org/
29) Provos, N.: Improving Host Security with System Call Policies, *Proc. 12th USENIX Security Symposium*, Washington, DC (2003).
30) Ramachandran, M., Smith, N., Wood, M., Garg, S., Stanley, J., Eduri, E., Rappoport, R., Chobotaro, A., Klotz, C. and Janz, L.: New Client Virtualization Usage Models Using Intel Virtualization Technology, *Intel Technology Journal*, Vol.10, No.3, pp.205–216 (2006).
31) Sailer, R., Jaeger, T., Valdez, E., Caceres, R., Perez, R., Berger, S., Griffin, J. and Doorn, L.: Building a MAC-based Security Architecture for the Xen Opensource Hypervisor, *Proc. 21st Annual Computer Security Applications Conference*, Tucson (2005).
32) Symantec: Norton AntiVirus. http://www.symantec.com/nav/nav_9xnt/
33) TrendMicro: TREND MICRO. http://www.trendmicro.com/en/home/us/personal.htm/
34) VMware: VMware ACE. http://www.vmware.com/products/ace/
35) Wagner, D. and Dean, D.: Intrusion Detection via Static Analysis, *Proc. 2001 IEEE Symposium on Security and Privacy*, Oakland (2001).

付　　録

A.1　セキュリティポリシ文法

PolicyFile	→	DefMacro ∗ `default` :DefAction PolOption∗ ModuleSpec∗
DefMacro	→	`includeMacro`(*path*)
DefAction	→	Action | `skip`
Action	→	`allow` | `deny`(*errno*) | `killProc`(*signum*) | `killThread`(*signum*) | `policyChange`(*policyfile*[, *logfile*]) | `ask`
PolOption	→	`execByPtracingProc` :PtAction | `signalMask`(*signums*) | `controlChild` :`detachProc`
PtAction	→	`detachProc` | `createNewMonitor`
ModuleSpec	→	ModuleName `default`:DefAction SysCallSpec*
ModuleName	→	`processMod` | `fileMod` | `networkMod` | `ipcMod` | `signalMod` | `fsMod` | `idMod`
		| `memoryMod` | `systemMod` | `timeMod` | `unimplementedMod`
SysCallSpec	→	*syscallname* `default`:DefAction ControlExpr*
ControlExpr	→	Cond∗ Action
Cond	→	ProcessCond | FileCond | NetworkCond | IdCond | MemoryCond
		| CurrIdCond | `argEq`(*argnum*, *value*) | Cond `and` Cond | Cond `or` Cond
ProcessCond	→	`cloneFlagsEq`(*cloneflags*) | `ptraceRequest`(*requests*)
FileCond	→	`fileEq`(*argnum*, *path*) | `filePrefixEq`(*argnum*, *pathprefix*) | `fileFlagsEq`(*fileflags*)
NetworkCond	→	`socketDomainEq`(*domain*) | `socketTypeEq`(*socktype*) | `socketProtocolEq`(*sockprot*)
		| `ipaddrEq`(*ipaddr*) | `netaddrEq`(*netaddr*) | `portEq`(*portnum*) | `unixsockEq`(*unixsock*) | `unixsockPrefixEq`(*unixsock*)
		| `ifindexEq`(*ifindex*) | `packetTypeEq`(*packettype*) | `sockoptEq`(*level*, *optname*)
IdCond	→	`uidEq`(*idnum*) | `gidEq`(*idnum*) | `euidEq`(*idnum*) | `egidEq`(*idnum*)
MemoryCond	→	`memProtEq`(*protections*) | `memFlagsEq`(*memflags*)
CurrIdCond	→	`uidEq`(*idnum*) | `gidEq`(*idnum*) | `euidEq`(*idnum*) | `egidEq`(*idnum*)

currIdCond：捕捉時の制御対象プログラムの ID に関する条件を記述.

`ifindexEq`, `packetTypeEq`：AF_PACKET に関する条件を記述.

`argEq`：*argnum* 番号目の引数に関する条件（値 *value*）を記述.

A.2　Apache（IA-32）に対するセキュリティポリシ（抜粋）

```
### apache.pol
includeMacro("asm-generic/fcntl.h")
includeMacro("linux/socket.h")
...
default   : allow
processMod   default: ask
   execve   default: deny(EPERM)
      fileEq(1,"demo-cgi.cgi") and currUidEq(33)
         policyChange("cgi.pol")
   ...
fileMod default: ask
   open  default: ask
      filePrefixEq(1,"/etc/apache2/") and fileFlagsEq(O_RDONLY)
         allow
      fileEq(1,"/etc/apache2/conf.d")
         and fileFlagsEq(O_NONBLOCK|O_LARGEFILE|O_DIRECTORY|O_NDELAY)
         allow
   poll  default: skip
   ...
networkMod   default: ask
   socket default: ask
      socketDomainEq(AF_INET)
        and socketTypeEq(SOCK_STREAM) allow
      ...
   bind    default: ask
      sockaddrFamilyEq(AF_INET) and portEq(80) allow
...
```

```
### cgi.pol（xxx.xxx.xxx.xxx は IP アドレス）
includeMacro("asm-generic/fcntl.h")
includeMacro("linux/socket.h")
includeMacro("asm-generic/mman.h")

default   : ask
fileMod   default: ask
   open   default: ask
      fileEq(1,"demo-cgi.cgi")
         and fileFlagsEq(O_LARGEFILE)
         and currUidEq(33) allow
   ...
networkMod   default: ask
   connect     default: ask
      socketDomainEq(AF_INET)
         and ipaddrEq(xxx.xxx.xxx.xxx)
         and portEq(53)
         and currUidEq(33) allow
   ...
memoryMod   default: ask
   mmap     default: ask
      memProtEq(PROT_READ|PROT_EXEC)
         and memFlagsEq(MAP_PRIVATE)
         and currUidEq(33) allow
   ...
timeMod   default: ask
   gettimeofday default: skip
...
```

(平成 20 年 7 月 18 日受付)
(平成 20 年 12 月 5 日採録)

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 年東京大学工学部計数工学科卒業．1977 年 MIT 計算機科学科博士課程修了．Ph.D. in Computer Science．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，政府情報科学技術委員会委員，内閣府総合規制改革会議委員，同教育分野主査等を歴任．現在，情報システム研究機構監事，（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．エンジンゼロワン文化戦略会議教育委員会委員，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．