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

2007－OS－106 (5)
2007／8／3

仮想マシンモニタによる仮想マシン内プロセスの制御

尾上浩一[†] 大山恵弘[††] 米澤明憲[†]

仮想マシンモニタ（VMM）はその上で稼働する仮想マシン (VM) を互いに強く隔離することができる．よって，VM の外でセキュリティシステムを動かすことにより，VM 内部の動作を安全に制御することができる．しかしながら，VM の外から得られる情報は低レベルであるため，VM の外で動くセキュリティシステムが制御対象の VM 内の OS 資源を制御することは簡単ではない．本論文ではゲスト OS の情報を利用し，制御対象の VM の外から VM 内で動作するプロセスを制御するシステムを提案する．提案システムでは，制御対象の VM 内のプロセスが発行したシステムコールを VMM が捕捉し，別の VM 内で動くプログラムがその実行をセキュリティポリシーに基づいて制御する．予備実験では，マイクロベンチマークとウェブサーバ Apache のベンチマークを用いて，提案システムのオーバーヘッドを測定した．

Controlling Processes in Virtual Machines by Virtual Machine Monitor

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

A virtual machine monitor (VMM) can provide strong isolation between virtual machines (VMs) running on top of it. Hence, users can achieve secure control of operations in a VM by running security systems outside the VM. However, security systems running outside the target VM obtain only low-level events and execution states. Therefore, it is not straightforward to enable the security systems to control the OS resource. In this paper, we propose a system which controls processes running in target VMs from the outside by using the knowledge of the OS running in the target VMs. In our system, VMM intercepts system calls issued by processes in the target VMs, and a program running in another VM controls the execution according to a security policy. We measured the overhead incurred by our system through preliminary experiments using microbenchmarks and a benchmark for the Apache web server.

1. はじめに

VMM による仮想環境の提供は，安全性の面からは仮想環境間が強く隔離されるという利点がある．たとえ攻撃者が仮想マシン（VM）内のサーバを乗っ取っても，VMM や他の VM を乗っ取ることは困難である．

しかし，VMM を用いて実行環境を隔離するだけで十分な安全性が確保されるわけではない．確かに，VMM を用いると，乗っ取られた VM やアプリケーション以外の部分に被害が及ぶことは防ぎやすい．しかし，乗っ取られた部分の悪意利用を防ぐことが難しい．たとえば VM 内で運用しているウェブサーバが乗っ取られた場合，そのサーバが持つ機密情報が攻撃者の手に渡りうる．また，そのサーバの権限を利用してデータの改竄や消去が行われる可能性もある．VMM を用いて隔離された実行環境を構築する場合でも，高い安全性を実現するには，他のセキュリティシステムを組み合わせて用いることが効果的である．

セキュリティシステムと VMM を組み合わせて用いる場合，まず考えられるのは，既存のセキュリティシステムを**ゲスト OS の中で**稼働させる方法である．この方法には，新たな仕組みを導入することなく実現可能であるという利点や，セキュリティシステムがゲスト OS 内の豊富な情報を利用できるという利点がある．しかし，この方法には，もし攻撃者がセキュリティシステムによる検知を迂回し，OS の管理者権限を取得した場合に，セキュリティシステムを停止・無力化されてしまうという欠点がある．

他の選択肢は，セキュリティシステムを**ゲスト OS の外で**稼働させる方法である．この方法には，たとえ攻撃者がゲスト OS の管理者権限を奪取しても，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という利点がある．一方，この方法には，セキュリティシステムが利用できる情報が，ゲスト OS と VM

† 東京大学
University of Tokyo
†† 電気通信大学
The University of Electro-Communications

との相互作用を中心とする低レベルなイベントの情報に限られるという欠点がある．ただし，低レベルの情報から高レベルの情報を復元することができれば，この欠点は解消される．

本論文では，VMM が捕捉可能なイベントの情報を利用して，ゲスト OS 内で動作するプロセスの振る舞いを制御し，安全性を向上させるシステムを提案する．提案システムでは，ゲスト OS 内で発行されたシステムコールを VMM が捕捉し，そのシステムコールの情報に基づいてプロセスの振る舞いを制御する．利用者はセキュリティポリシーを通じて，どの VM 内のどのプロセスを制御対象とするか，また，そのプロセスのシステムコールの実行をどのように制御するかを指示する．制御対象の VM 内のプロセスを制御するために必要な情報（ゲスト OS が利用するデータ構造や関連するシンボル情報など）はゲスト OS イメージのコンパイル時に生成し，システムの利用時に VMM に与える．現在の実装では，ゲスト OS が Linux であることを仮定している．我々は VMM として Xen [5] を利用して提案システムの設計および実装を行い，予備評価を行った．

本論文は以下のように構成される．2 章と 3 章のそれぞれで提案システムの設計と実装について述べる．4 章で提案システムの予備評価について述べる．5 章で関連研究について述べる．6 章でまとめと今後の課題について述べる．

2. 提案システム

2.1 基本構成

VMM を利用した VM 内部のプロセスを制御するシステムの構成を**図 1** に示す．提案システムでは，プロセスの実行制御を行うプログラムを特別な VM 上で動かす．本論文では上記のプログラムを**制御プログラム**，上記の VM を**制御 VM** と呼ぶ．また，内部で動作するプロセスの実行が制御される VM を**制御対象 VM**，実行が制御されるプロセスを**制御対象プロセス**と呼ぶ．

提案システムでは，制御 VM はゲスト OS カーネルに依存した動作制御を行う．動作制御は，制御対象 VM 内のプロセスが発行したシステムコールの実行を制御することによって実現する．どのように制御するかは，利用者がセキュリティポリシーによってシステムに与える．

提案システムでは制御対象プロセスを制御するために以下の二つのコマンドを提供する．

vps: 利用者は制御 VM 内でこのコマンドを実行することにより，制御対象 VM 内のプロセスの情報を取得する．その情報には，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 などが含まれる．直感的には，このコマンドは UNIX の ps コマンドを VM の外から発行できるようにしたものと考えることができる．

vcntl: 利用者は制御 VM 内でこのコマンドを実行することにより，制御対象 VM 内のプロセスの制御操作を行う．制御開始前の設定を行うために，VMM が利用するゲスト OS に関する情報とセキュリティポリシーが記述された各ファイルを与えるコマンドを実行する．プロセス制御開始時には，制御対象の VM ID，制御対象となるプロセスの ID のリストまたはプログラムのパス名を与えるコマンドを実行する．

2.2 セキュリティポリシー

現在のシステムにおけるセキュリティポリシーの文法と例をそれぞれ**図 2**，**図 3** に示す．システムコールの番号と引数の情報を利用して実行を制御することが可能になっている．また，図 2 の対応処理 (Action) は，システムコールの実行の許可，実行の失敗，制御対象プロセスの強制終了，適用するポリシーファイルの変更を示している．

図 3 のセキュリティポリシーの場合，read，execve 以外の実行は失敗し，返り値として-1 を返す．さらにファイル`/etc/passwd` とディレクトリ`/etc/cron.d` の下のファイルのオープンに失敗し，他のファイルのオープンは許可する．execve が実行された場合には，`/usr/bin/wserver` が実行されたときに限り `/usr/bin/wserver` 用のポリシーに切り替えて実行制御を継続する．その他の場合は強制終了する．

システムコール捕捉を利用したプロセス制御には以下の利点がある．まず，Linux のシステムコールの数は高々約 300 であり，人間がすべてを把握することが可能な範囲にある．また，システムコールはプロセス操作，ファイルやネットワークなどの計算資源操作に分類可能で，直感的な理解がしやすい．さらに，ユーザアプリケーションが計算資源を操作する際にはシステムコールが発行されるため，システムコールを制御することにより，悪意あるユーザアプリケーションがシステムコールを迂回して計算資源を操作することは困難である．

システムコール捕捉に基づくセキュリティシステムについては，過去に多くの研究 [6, 9, 11] があるので，今後その成果を利用してさらにシステムを洗練させることができるという利点もある．

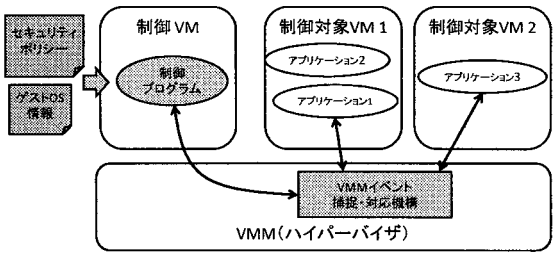

図 1　提案システムの構成

PolicyFile	→	`default` :Action `traceChild` :YesNo SysCallSpec∗
YesNo	→	`yes` | `no`
SysCallSpec	→	SysCallName `default`:Action ControlExpr*
ControlExpr	→	Condition∗ Action
Condition	→	FileCond | NetworkCond | Condition `and` Condition | Condition `or` Condition
FileCond	→	`fileEq`(*argnum*, *path*) | `filePrefix`(*argnum*, *pathPrefix*)
NetworkCond	→	`ip`(*ipaddr*) | `port`(*portnum*) | `protocol`(Protocol)
Protocol	→	`tcp` | `udp`
Action	→	`allow` | `deny`(*retnum*) | `killProc` | `policyChange`(*policyfile*)

図 2　提案システムにおけるセキュリティポリシー文法（抜粋）

```
default : deny(-1)
traceChild : yes
open    default : allow
        fileEq(1,"/etc/passwd") or filePrefix(1,"/etc/cron.d") deny(-1)
execve  default : killProc
        fileEq(1,/usr/bin/wserver) policyChange(wserver.pol)
```

図 3　提案システムにおけるセキュリティポリシーの記述例

2.3 特　　長

提案システム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 内の制御対象外のプロセスを乗っ取っても，仮想マシンモニタの VM 間の隔離により，制御 VM およびモニタプログラムを直接攻撃することはできない．

プロセス単位での実行制御が可能　制御対象プロセスが不正処理や異常動作をしたときには，制御対象プロセスのみに対応処理が適用され，制御対象外のプロセスの実行状態は保たれる．一方，制御単位が VM である場合，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セスも強制終了されてしまうなどの欠点がある．

複数 VM 内のプロセスを統一的に制御可能　同一プログラムがひとつの VMM 上の複数の 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プロセスを，共通のセキュリティポリシーによって制御できる可能性がある．たとえば，ウェブサーバ Apache やメールサーバ Sendmail に対するセキュリティポリシーを用意し，各 VM 内で動作するこれらのサーバのプロセスへ適用する．

VMM 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，VMM の管理者が制御対象 VM 内のプロセスを制御することにより，各 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．提案システムにより，

```
[dom0] # vcntl setup 1 guest.txt syscall.txt
[dom0] # vps 1 stat.txt
[dom0] # vcntl start 1 proclist.txt policy.txt
```

図 4　提案システムの利用の流れ

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．

2.4 利 用 例

提案システムにより制御対象 VM（VM ID:1）内のプロセスを制御する例を**図 4** に示す．1 行目で利用者は制御対象 VM 内のプロセスに必要な設定を行っている．現在の提案システムではゲスト OS に関する情報（`guest.txt`）とシステムコール捕捉に関する情報（`syscall.txt`）を用いる．2 行目では制御対象 VM のプロセスリストを `stat.txt` に出力している．利用者はその中からプロセスを選択し，`proclist.txt` に書き込む．3 行目ではセキュリティポリシーを記述したファイル（`policy.txt`）を与え，プロセスの制御を開始させている．

3. 実　　装

3.1 プロセス実行制御機構

VMM として疑似仮想化技術を利用した Xen 3.0.3.0，ゲスト OS として Linux 2.6.16.19 を利用し，提案システムの実装を行った．

VMM は，システムコールの開始・終了処理部分を捕捉し，制御対象 VM 内のプロセスの実行を制御する．なお，捕捉機構の導入に伴うゲスト OS のソースコードの修正は必要ない．

2.1 節で述べたように，制御 VM は二つのコマンドを提供する．

3.1.1 プロセス情報の取得

vps は VMM に対する処理要求を発行し，指定された VM ID の制御対象 VM のプロセス情報を VMM を通して取得する．指定された VM ID の制御対象 VM のプロセス情報を VMM が取得する手続きは以下の通りである．まず，もし制御対象 VM が実行中であった場合には実行を停止させ，仮想 CPU が保持している実行状態を用いて，制御対象 VM の仮想アドレス空間に切り換える．そして，制御対象 VM が保持しているレジスタの値を用いてプロセスリストを取得する．その後，必要な場合には仮想アドレス空間を制御 VM に対応するものに戻し，取得したプロセスの実行状態を制御 VM に通知する．

3.1.2 制御対象プロセスの登録

vcntl コマンドが VMM に対する処理要求を発行すると，VMM が制御対象プロセス管理用リストに，プロセス ID またはプログラムパス名を追加する．制御対象 VM で execve が呼び出されるたびに，VMM はそれが制御すべきプログラムパス名かどうかを検査し，そうである場合にはそのプロセスの ID を制御対象プロセス管理用リストに追加する．新たに生成したプロセスに対しての制御も行うようにセキュリティポリシーファイルに記述されていた場合にも制御対象プロセス管理用リストに追加する．

3.1.3 システムコールの実行制御

制御対象プロセスによって発行されたシステムコールの実行を制御する流れは次の通りである．制御対象 VM 上のプロセスがシステムコールを発行したときに，VMM はそれが制御対象プロセスか制御対象外プロセスかを調べる．制御対象外プロセスである場合には，VMM は何もせず，制御対象 VM 内でそのシステムコールの実行を継続させる．制御対象プロセスである場合には，VMM は制御対象 VM の仮想 CPU の実行状態，そのプロセスのシステムコール番号と引数を読み出す．その後，VMM はシステムコールが呼び出されたことを制御 VM へ通知する．制御 VM はセキュリティポリシーに基づき，発行されたシステムコールに対する制御方法を決定し VMM に通知する．制御対象プロセスは制御プログラムにより制御方法の通知があるまで停止させられる．VMM と制御 VM 間での制御 VM の決定に従い，VMM が制御対象プロセスを再開する．

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，対象プロセスのみを停止するようにしている．これは複数の CPU が利用可能で，システムコール検査に要する時間が長いときに，制御対象外のプロセスの実行を継続可能とするためである．

3.2 VM の外からのプロセス情報の取得

Linux において，主なプロセス情報はカーネル内の `task_struct` 構造体に保持されている．`task_struct` 構造体の各メンバを正しく参照するためには，`task_struct` 構造体のデータレイアウトを知る必要がある．このデータレイアウトは，OS カーネルのバージョンと OS イメージ作成時の設定に依存する．このため，提案手法では，ゲスト OS イメージの生成時に `task_struct` 構造体のデータレイアウトに関する情報を同時に生成する．また，ゲスト OS カーネルに依存するシステムコール番号や数に関する情報もこのときに生成する．利用者は制御対象 VM の制御開始前に

この情報を与える．

VMM は，制御対象 VM 内のプロセス情報を得る際に，task_struct 構造体へのポインタを取得する．Linux 2.6 以降では task_struct 構造体へのポインタが thread_info 構造体のメンバとなっている．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計算し，そこから task_struct 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，GS レジスタから thread_info 構造体へのポインタが取得可能である．

Xen では VMM のメモリ領域はすべての VM で共有されており，システムコールが発行されて VMM に処理が切り換えられたときに仮想アドレス空間を変更しない．このため，ゲスト OS カーネルが管理する task_struct 構造体に直接アクセスすることが可能である．Linux カーネルではプロセス情報のようなカーネルのメモリ領域にアクセスする場合にはページフォルトは生じない．

3.3 システムコール捕捉機構

3.3.1 Linux におけるシステムコール手続き

Linux 2.6 以降では，IA-32 アーキテクチャ用のコードにおいては，CPU のバージョンに従い，ソフトウェア割り込みを用いた方法 (INT 0x80 の実行) と SYSENTER 命令を用いた方法でシステムコール呼び出しをサポートしている．システムコール終了後には SYSEXIT 命令を用いてユーザレベルに移行することも可能である．

SYSENTER は Pentiun II 以降の IA-32 アーキテクチャが提供している命令であり，SYSENTER の動作を設定するための値は Model Specific Register (MSR) が保持している．MSR への書き込み命令は特権命令である．SYSENTER 命令が実行されると，OS カーネルが事前に MSR に設定しておいた値が，インストラクションポインタやスタックポインタとしてレジスタにセットされ，特権レベルが 0 に変更される．

3.3.2 Xen におけるシステムコール手続き

疑似仮想化技術を利用した Xen 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部分へ遷移する．これを実現するために，Xen はゲスト OS のソースコードの割り込みディスクリプタテーブルの設定操作を修正している．

3.3.3 提案システムにおけるシステムコール捕捉

我々は疑似仮想化技術を利用する Xen の VMM を修正し，制御対象 VM で発行されたシステムコールの呼び出しと終了などのイベントを提案システムが捕捉できるようにした．

ソフトウェア割り込みを利用したシステムコール呼び出しに対応するために，提案システムでは，ゲスト OS カーネルのバイナリ書き換えを行う仕組みを導入している．vcntl が制御対象 VM 内のゲスト OS カーネルのシステムコールの実行を行う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．システムコールの実行を行うバイナリコードの仮想アドレスはゲスト OS カーネルイメージ生成時に取得する．書き換え後の HLT 命令の実行は VMM によって捕捉される．捕捉後，VMM はシステムコール制御処理を行い，システムコールの実行を継続する場合には書き換え前の命令をエミュレートした後，ゲスト OS カーネルのコードに制御を移す．

SYSENTER 命令を利用したシステムコール呼び出しに対応するために，提案システムでは，VM による SYSENTER 命令を VMM が捕捉できるように VMM が起動時に MSR の値を設定する．各ゲスト OS カーネルが SYSENTER 命令に関連する値を MSR に書き込もうとしたときには，実際の MSR には値を書き込まず，VMM がその値を保存する．ユーザプロセス（特権レベルが 3 のプロセス）のうち制御対象となっているもが SYSENTER 命令を実行したときには，VMM が MSR に書きこんでおいた捕捉用のコードに制御が移る．VMM はシステムコールの制御処理を行った後，特権レベルを 1 （Xen におけるゲスト OS カーネルの特権レベル）に変更し，あらかじめ保存しておいた，ゲスト OS カーネル内のシステムコール処理用コードに制御を移す．

SYSEXIT 命令は，特権レベル 1 で動作するゲスト OS カーネルが実行した場合に例外が発生するため，特別な修正なしに VMM が捕捉可能である．

システムコールの終了処理を行うバイナリコード部分も VMM が捕捉する必要がある．これは fork や accept のようなシステムコール終了後の状態を検査することが必要なシステムコールが存在するためである．終了処理の捕捉は，ソフトウェア割り込みを利用したシステムコール呼び出しへの対応と同様，制御対象 VM 内のゲスト OS カーネルのバイナリ書き換えを利用する．

3.4 システムコール引数の取得

システムコール制御のために制御 VM はシステムコール引数を利用する．制御 VM が利用する各引数の値は VMM によりシステムコール捕捉時に保存される．システムコール引数にはユーザ空間の仮想アドレスを表しているものも含まれる．たとえば，open

システムコールのファイル名やconnectシステムコールのネットワークアドレス情報である．task_struct構造体のようなプロセス情報を保存しているメモリ領域とは異なり，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．

システムコール引数がユーザ空間の仮想アドレスを表している場合，VMMが現在実行中のゲストOSのページテーブルを参照し，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ポインタ参照に伴いページフォルトが生じるため，VMMは制御VMにシステムコールの発行を通知する前にゲストOSにページフォルト処理を実行させる．ゲストOSがページフォルト処理を完了した後，VMMに処理が戻り，VMMは該当するユーザ空間のメモリ領域を参照する．

3.5 制御対象プロセスへの対応処理

図2で示した制御対象プロセスへの対応処理(Action)について述べる．

提案システムではシステムコールの実行制御が，制御対象VMのゲストOSのシステムコールの呼び出し規約に依存する．システムコールを失敗させる処理(deny(*retnum*))では，制御対象プロセスが呼び出すシステムコールをgetpidに変更する．getpidの終了処理時に*retnum*で指定された返り値を設定する．制御対象プロセスを強制終了する処理(killProc)では，制御対象プロセスに対応するtask_struct構造体から到達可能なシグナル情報を変更する．

4. 予備評価

現在の提案システムに対し，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実験を行った．VMM上では制御VMと1つの制御対象VMを稼働させる．制御VMと制御対象VMは2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．

4.1 マイクロベンチマーク

制御対象VMが発行するシステムコール制御に伴うオーバーヘッド（VM-VM）の計測するために，制御対象VM内で以下の3つのシステムコール捕捉処理を10000回繰り返しそれに要する時間を計測した．SYSENTER命令(SYSENTER)とソフトウェア割り込み(INT)のそれぞれに要する時間を計測した．実験環境はCPU Pentium 4 3.0 GHz（Hyper-Threading有効），1 GBメモリである．時間計測にはgettimeofdayを利用した．

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 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．ページフォールトの発生有無の検査，ファイル名の取得が必要である．

fork,waitpid,exit_group 制御対象のプロセス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．新たに生成されたプロセスを制御対象に追加する処理が必要になる．

VMMによるシステムコール捕捉のオーバーヘッドを計測するため，VMMが制御対象VMが発行したシステムコールを捕捉後制御VMに通知せず，システムコールを再開させた場合に要する時間(VM-VMM)の計測も行った．比較のために，上記の3つのシステムコール捕捉処理をLinux 2.6.20 (Linux)，Xen上で各々実行した．

実験結果を**図5**に示す．図中のグラフは一連のシステムコール処理を10000回繰り返したときの時間（ミリ秒）を表し，小さい方がシステムとしては有用である．実験結果からSYSENTERを利用したシステムコール捕捉機構の方がソフトウェア割り込みを利用した場合よりもオーバーヘッドが小さくなっていることがわかる．

4.2 アプリケーションベンチマーク

制御対象VM内部で動作するアプリケーションが発行するシステムコールを捕捉することによって生じるオーバーヘッドを計測するために，ApacheBench 2.0.40を用いた．

制御対象VM内でウェブサーバApache 2.0.54を動作させ，同一LAN内の異なる計算機上のApacheBenchから1 KBのファイル要求を行う．要求数は1，2，32，256，1024と変化させた．提案システムを動作させる計算環境は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．ApacheBenchを実行する計算環境は，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る．

制御VMから制御対象のApacheのプロセスを制御するために，Apacheのプロセス群をvpsコマンドを用いて取得する．この実験で制御開始時に指定するApacheに関連するプロセス数は約60であった．制御VMによるシステムコール捕捉後の対処方法はすべて許可ということにした．この実験においてはApache

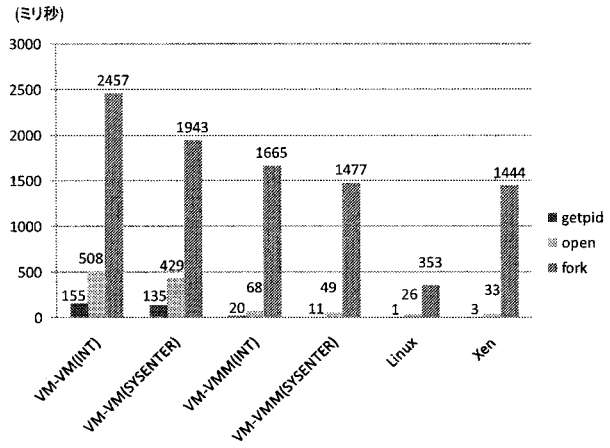

図 5 システムコールの実行時間

要求数	1	2	32	256	1024
提案システム	0.6	1.1	17.4	142.8	572.9
Xen	0.1	0.2	0.3	0.2	0.8
VMM による制御	0.1	0.2	0.3	0.3	0.9

表 1 1 KB ファイルの取得要求に要する時間（秒）

が発行するソフトウェア割り込みを利用したシステムコールを捕捉する．

比較のために，システムコール捕捉を行わない Xen 上で動作させた場合と，VMM がシステムコール捕捉後に制御 VM に通知を行わず Apache への要求処理を継続させた場合の処理時間も測定した．

実験結果を表 1 に示す．この実験においてはすべてのファイル取得要求は成功した．表 1 中の提案システムと VMM による制御を比較することにより，制御 VM・制御対象 VM 間の通知機構の設計・実装を洗練させていくことによってオーバーヘッドを削減させていきたい．

5. 関連研究

VMM 上で動作する VM の安全性を高めるための研究は数多く提案されている．sHype [12] は，VMM 上で動作する複数の VM 間に Chinese Wall や Type Enforcement のようなセキュリティポリシーを適用できるようにするシステムである．sHype により，複数の VM 間を強く隔離することができる．VMware ACE [2] は，VM のイメージとセキュリティポリシーを組にしたパッケージを作成，配布できるようにし，VM の安全な実行を可能にするシステムである．セキュリティポリシーには VM の使用期限，デバイスやネットワークアクセスに関する制御方法を記述する．NetTop [3] は隔離された実行環境を構築するためのシステムである．VMM として VMware，ホスト OS として SELinux [1] を用いている．SELinux を利用した強制アクセス制御，情報流制御により，複数の VM を強く隔離することができる．これらのシステムでは制御が VM 単位であるのに対し，提案システムは VM 内のプロセス単位の制御を行う．

提案システムのように VM の外から VM 内の動作を制御する研究として Livewire [8] がある．Livewire では，制御対象となっている VM 内の資源の周期的な改竄検査と特定の操作に対する制御を異なる VM から行う．制御対象の VM 内の動作を制御するためにゲスト OS 情報を利用している．Livewire では VMM が捕捉する操作は書き込み禁止のメモリ領域へのアクセスとネットワークデバイスへのアクセスであり，本研究では制御対象 VM 内のプロセスが発行するシステムコールの実行を捕捉する．

IntroVirt [10] は，VM 内で動作する既知の脆弱性を持つプログラムを，Predicate による仕様に従って別の VM から制御することを可能にするシステムである．IntroVirt による制御対象 VM 内の制御は脆弱性を含むコードの捕捉に基づいているが，本研究では特定の制御対象プロセスのシステムコール実行に基づいた制御を行う．

文献 [4] のシステムは制御対象 VM を honeypot として利用し，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．このシステムでは制御対象 VM 内のゲスト OS カーネルが実行した特定操作を捕捉するための機構 (Sensor) を導入している．Sensor は，ソケット操作やファイル操作などを捕捉し VMM に

通知する．このシステムでは監視処理がすべてVMMで行われるが，本研究ではシステムコールに基づく対応処理の決定は制御VMで行う．

これまで，システムコール捕捉に基づいたプロセス，ファイル・ネットワークなどの計算機資源に基づいた，侵入検知システム [7,13] やサンドボックスシステム [9,11] 等のセキュリティシステムが盛んに提案・開発されてきている．本研究の提案システムの設計はカーネルレベルでシステムコールを捕捉し，ユーザレベルで制御方法を決定するシステム [9,11] と同じ設計方針で，VMMでシステムコールを捕捉し，制御VM内の制御プログラムで制御方法の決定を行う．これらの同一OS内の制御プログラムによるプロセス制御と異なり，制御対象プロセスから提案システムにおける制御VM内の制御プログラムを直接攻撃することはできない．

6. まとめと今後の課題

本論文では，制御対象となるVM内のプロセスが発行するシステムコールを異なるVMから制御し，制御対象VMの安全性を向上させるシステムを提案した．提案システムはVM内で動作するゲストOSカーネル情報を利用し，セキュリティポリシーによりプロセス制御を行う．我々はVMMとしてXenを，ゲストOSとしてLinuxを用いて提案システムを実装した．予備実験としてシステムコールに関するマイクロベンチマークとApacheを用いたアプリケーションベンチマークを行った．

今後の課題として，セキュリティポリシーの記述者の手間と負担を軽減するために，セキュリティポリシーの抽象度を上げていきたい．また，今回はApacheの動作を制御する実験を行ったが，今後はより広範囲のアプリケーションに提案システムを適用していきたい．現在のところ，提案システムにおける制御対象プロセスはユーザレベルアプリケーションであり，ゲストOSカーネルのデータを改竄する攻撃には対処できない．これへ対応することも今後の課題である．

参 考 文 献

1) *Security-Enhanced Linux.* `http://www.nsa.gov/selinux/`.
2) *VMware ACE.* `http://www.vmware.com/products/ace/`.
3) *NetTop*, 2004. `http://www.hp.com/hpinfo/newsroom/press_kits/2004/security/ps_nettopbrochure.pdf`.
4) K. Asrigo, L. Litty, and D. Lie. Using VMM-Based Sensors to Monitor Honeypots. In *Proceedings of the 2nd International Conference on Virtual Execution Environments*, Ottawa, June 2006.
5) P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, New York, October 2003.
6) S. Bhatkar, A. Chaturvedi, and R. Sekar. Dataflow Anomaly Detection. In *Proceedings of the 2006 USENIX Symposium on Security and Privacy*, Oakland, May 2006.
7) S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A Sense of Self for Unix Processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, Oakland, May 1996.
8) T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *Proceedings of the 10th Annual Network and Distributed Systems Security Symposium*, San Diego, February 2003.
9) I. Goldberg, D. Wagner, R. Thomas, and E.Brewer. A Secure Environment for Untrusted Helper Applications. In *Proceedings of the 6th USENIX Security Symposium*, San Jose, July 1996.
10) A. Joshi, S. King, G. Dunlap, and P. Chen. Detecting Past and Present Intrusions through Vulnerability-Specific Predicates. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles*, Brighton, October 2005.
11) N. Provos. Improving Host Security with System Call Policies. In *Proceedings of the 12th USENIX Security Symposium*, Washington, DC, August 2003.
12) R. Sailer, T. Jaeger, E. Valdez, R. Caceres, R. Perez, S. Berger, J. Griffin, and L. Doorn. Building a MAC-based Security Architecture for the Xen Opensource Hypervisor. In *Proceedings of the 21st Annual Computer Security Applications Conference*, Tucson, December 2005.
13) D. Wagner and D. Dean. Intrusion Detection via Static Analysis. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, Oakland, May 2001.