Computer Security Symposium 2019
21 - 24 October 2019

重要サービスの動作不可視化手法における仮想計算機停止の回避

奥田 勇喜[1,a)]　佐藤 将也[1]　谷口 秀夫[1]

概要：セキュリティソフトウェアやログ収集ソフトウェアなどの重要サービスへの攻撃を回避するために，重要サービスの動作を攻撃者から不可視化する手法が提案されている．この手法は，保護対象の仮想計算機（VM）上で動作する重要サービスが発行したシステムコールを仮想計算機モニタにより検知し，別 VM 上の代理プロセスによりシステムコールを代理実行する．しかし，この手法は，システムコールを代理実行する間に保護対象 VM を停止させてしまう問題がある．そこで，この VM の停止を回避する手法を述べる．具体的には，システムコールを代理実行する間に，あたかも重要サービスが `sched_yield()` システムコールを発行したかのように保護対象 VM を動作させることで，重要サービス以外のプロセスに動作する機会を与え，VM の停止を回避する．

キーワード：重要サービス，仮想計算機，攻撃回避

Avoidance of Virtual Machine Pausing on Behavior-Hiding Method for Essential Services

Yuuki Okuda[1,a)]　Masaya Sato[1]　Hideo Taniguchi[1]

Abstract: To avoid the attacks on essential services such as security software or logging programs, we proposed a method to make the behavior of the essential services invisible to attackers. The method detects system calls of essential services on a protection target virtual machine (VM) by using VM monitor and proxies them by the proxy process on another VM. However, the protection target VM pauses while proxying a system call. In this paper, we propose a method for avoiding the VM pausing. Specifically, while a system call is proxying, the VMM operates the protection target VM as if the essential service invoked a `sched_yield()` system call. As a result, processes other than the essential service are given the opportunity to operate and the VM pausing can be avoided.

Keywords: Essential Service, Virtual Machine, Attack Avoidance

1. はじめに

計算機上では，攻撃の防止や攻撃による被害を軽減するためにサービスが提供されている．例えば，セキュリティソフトウェアは，悪意のあるソフトウェア（以降，マルウェア）からの攻撃を防止する．また，ログ収集ソフトウェアは，攻撃の検知や被害の分析のためにプログラムの動作ログを収集する．以降，これらのサービスを重要サービスと呼ぶ．重要サービスは，攻撃者にとって不都合な存在であるため，攻撃の対象となり，無効化される可能性がある．文献 [1] では，いくつかのセキュリティソフトウェアの脆弱性を利用した攻撃手法が報告されている．また，ルートキットの一種である Adore-ng [2] は，マルウェアの侵入ログを隠蔽するために，ログ収集ソフトウェアである syslog におけるログ受信を妨害する．重要サービスが無効化されると，被害の軽減や原因の調査が困難になるため，重要

[1] 岡山大学 大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University
[a)] okuda2@swlab.cs.okayama-u.ac.jp

サービスを保護することは重要である．

重要サービスを保護する手法として以下がある．ANSS [3] は，攻撃者によるセキュリティソフトウェアの終了を防止するために，仮想計算機モニタ (Virtual Machine Monitor，以降，VMM）を用いて仮想計算機（Virtual Machine，以降，VM）を監視し，セキュリティソフトウェアを終了させるようなシステムコールが発行された時に，これを無効化する．また，Process Out-grafting [4] は，脆弱性のある VM 上で動作するセキュリティソフトウェアを信頼できる VM に移動させ，離れた VM からのマルウェア解析を可能にする．上記に対し，重要サービスの特定を困難化することができれば，重要サービスへの攻撃はより困難にでき，攻撃を回避できる可能性を高くできる．我々は，攻撃者による重要サービスの特定を困難化することに着目し，重要サービスの動作を不可視化する手法を提案した [5][6]．この手法は，保護対象 VM で発行されたシステムコールを VMM により捕捉し，重要プロセスが発行したファイル操作や通信に関するシステムコールを代理 VM 上に用意した代理プロセスにより代理実行する．これにより，当該システムコールは保護対象 VM 上で実行されず，保護対象 VM 内では不可視化されるため，ファイル操作内容や通信内容をもとにした重要サービスの存在の特定を困難化できる．しかし，この手法は，システムコールを代理実行する間に保護対象 VM を停止させてしまう問題がある．このため，代理実行中の保護対象 VM の停止を回避する必要がある．

本稿では，代理実行する間の保護対象 VM の停止を回避する手法を述べる．本手法は，代理プロセスによる代理実行処理の間に保護対象 VM に制御を戻して重要プロセス以外のプロセスに動作する機会を与え，代理プロセスによる代理実行後に結果を重要プロセスへ返却する．これにより．代理実行する間の保護対象 VM の停止を回避できる．また，評価では，保護対象 VM の停止回避の効果を示す．

2. 重要サービスの動作不可視化手法

2.1 基本構造

我々は，重要プロセスが行うファイル操作や通信に関する情報をもとにした重要サービスの特定を困難化するために，重要サービスの動作不可視化手法（以降，既存手法）を提案した．既存手法は，保護対象 VM で発行されたシステムコールを VMM により捕捉し，重要プロセスが発行したファイル操作や通信に関するシステムコールを代理 VM 上に用意した代理プロセスにより代理実行する．既存手法の基本構造を**図 1** に示す．既存手法では，VMM として Xen [7] を使用し，VM 上で動作するオペレーティングシステム（Operating System，以降，OS）として Linux を用いる．VMM 上では，重要サービスが動作する保護対象 VM と代理プロセスが動作する代理 VM が動作する．保護対象 VM は，Intel VT-x を用いて完全仮想化され，`syscall` 命令によりシステムコールが発行される環境を想定する．

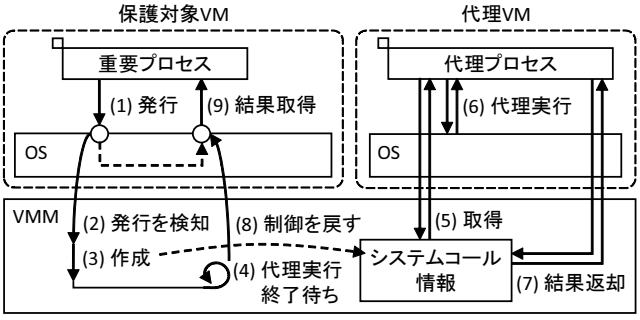

図 1 システムコールを代理実行する手法の基本構造
Fig. 1 Design of the system call proxy.

以下に，既存手法における代理実行の処理流れを示す．

(1) 重要プロセスがファイル操作や通信に関するシステムコールを発行する．
(2) VMM は，ハードウェアブレークポイントを利用して保護対象 VM 上でのシステムコールの発行を検知する．
(3) 検知したシステムコールがファイル操作や通信操作である場合，VMM はシステムコール情報を作成する．そうでない場合は，保護対象 VM に制御を戻す．システムコール情報は，代理実行するために代理プロセスに渡す情報であり，システムコールの番号，引数，およびバッファからなる．システムコールの番号と引数は，保護対象 VM のレジスタから取得できる．引数がポインタである場合は，ポインタが指す領域を保護対象 VM のメモリからバッファにコピーする．
(4) VMM は，代理プロセスによる代理実行が終了するのを待つ．このとき，CPU はビジーループして結果返却の有無を確認する．
(5) 代理 VM 上の代理プロセスは，定期的にシステムコール情報の有無を確認し，VMM からシステムコール情報を取得する．
(6) 代理プロセスは，取得したシステムコール情報をもとに，システムコールを代理実行する．
(7) 代理実行後，代理プロセスは，代理実行の結果を VMM に返却する．
(8) VMM は，代理実行の結果を保護対象 VM のレジスタやメモリに格納し，命令ポインタを操作してシステムコール終了処理に制御を戻す．
(9) 保護対象 VM では，システムコール終了処理が実行される．これにより，重要プロセスは，代理実行されたシステムコールの結果を取得する．

上記の処理流れにより，重要プロセスが発行したシステムコールを代理 VM 上に用意した代理プロセスにより代理実行することで，重要サービスが行うファイル操作や通信を保護対象 VM 上の攻撃者から不可視化できる．

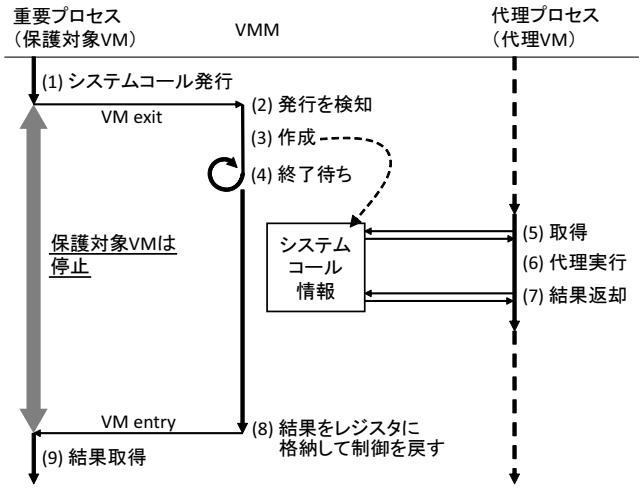

図 2　システムコールを代理実行する手法の問題点
Fig. 2　Problem of the system call proxy.

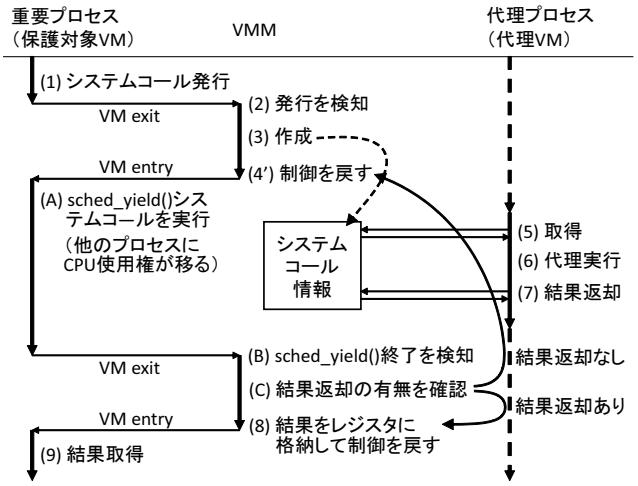

図 3　提案手法の処理流れ
Fig. 3　Flow of the proposed method.

2.2　問題点

既存手法には，代理実行する間，保護対象 VM に CPU が割り当てられないという問題がある．この問題について，図 2 に示す．VMM は，重要プロセスが発行したシステムコールを検知 (2) してから，代理実行の結果を重要プロセスに返却する (8) まで，保護対象 VM の CPU を使用して動作する．このため，保護対象 VM は，VMM が動作する間は停止する．特に，保護対象 VM の仮想 CPU が 1 つの場合，保護対象 VM 全体が停止する．保護対象 VM を効率的に動作させるためには，代理実行している間に保護対象 VM に CPU を割り当てて，保護対象 VM の停止を回避する必要がある．

3.　仮想計算機停止の回避

3.1　要件

保護対象 VM の停止を回避するため要件を以下に示す．

（要件 1）保護対象 VM に CPU を割り当てて，重要プロセス以外のプロセス（以降，他プロセス）に動作する機会を与えること

（要件 2）重要プロセスが代理実行の結果を受け取れること

保護対象 VM の停止を回避するためには，保護対象 VM に CPU を割り当てて，他プロセスに動作する機会を与える（要件 1）必要がある．また，保護対象 VM に CPU を割り当てると，VMM は代理実行の終了を待機できないため，代理実行の結果が代理プロセスから VMM に返却された際，代理実行の結果を重要プロセスに返却できない，このため，重要プロセスが代理実行の結果を受け取れるようにする（要件 2）必要がある．

3.2　対処

(要件 1) の対処として，保護対象 VM に制御を戻し，`sched_yield()` システムコールが実行されるようにする．`sched_yield()` システムコールは，他のプロセスに CPU 使用権を譲る機能をもつ．これにより，保護対象 VM に制御を戻したときに，重要プロセスが `sched_yield()` システムコールを発行したことになるため，他プロセスに動作する機会を与えることができる．このとき，保護対象 VM 上では，OS がもつスケジュール機能の呼び出しにより，通常のスケジューリング時と同様にプロセス切り替えが行われるため，他プロセスが動作しても問題ない．保護対象 VM で `sched_yield()` システムコールを実行するためには，保護対象 VM に処理を戻す前に，システムコール番号が格納されている RAX レジスタの内容を `sched_yield()` のシステムコール番号（Linux 3.2.0 では 24）に変更する．また，`sched_yield()` システムコールは引数を持たないため，保護対象 VM に加える変更を最小限にできる．

(要件 2) の対処として，`sched_yield()` システムコールの終了時に代理実行の結果の有無を確認する処理を追加する．具体的には，`sched_yield()` システムコールの終了処理を VMM により捕捉し，代理実行の結果が代理プロセスから VMM に返却されている場合は，結果を保護対象 VM のレジスタやメモリに格納し，保護対象 VM に制御を戻す．代理実行の結果が返却されていない場合は，保護対象 VM で再度 `sched_yield()` システムコールを実行させる．これにより，重要プロセスに代理実行の結果を返却できる．また，代理プロセスから代理実行の結果が返却されるまで，他プロセスに動作する機会を与えることができる．

3.3　処理流れ

提案手法の処理流れを図 3 に示し，以下で説明する．既存手法の (4) を変更し，(A)，(B)，および (C) を追加した．

(4′) VMM は，システムコール番号が格納されている RAX レジスタの値を `sched_yield()` のシステムコール番

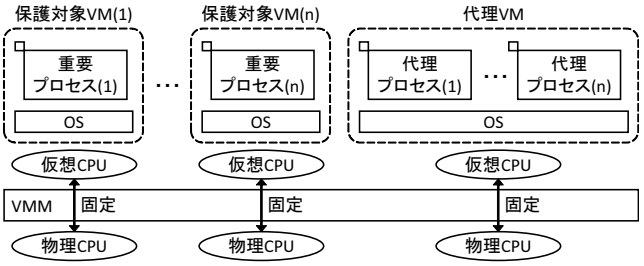

図 4　評価モデル
Fig. 4　Model of the evaluation.

号に置き換えた後，保護対象 VM に制御を戻す．

(A) 保護対象 VM では，重要プロセスが `sched_yield()` システムコールを発行したことになり，他プロセスに CPU 使用権が移る．これにより，代理プロセスが代理実行する間，他プロセスに動作する機会を与えることができる．

(B) 重要プロセスが発行した `sched_yield()` システムコールの終了処理を VMM により検知する．

(C) 代理プロセスからの結果返却の有無を確認する．既に結果が返却されている場合，(8) に移行する．まだ返却されていない場合，再度システムコールを `sched_yield()` に置き換えた後，命令ポインタを操作してシステムコール開始処理に制御を戻す．

上記の処理により，VMM によりシステムコール情報を作成してから代理プロセスにより代理実行の結果を返却されるまでの間，保護対象 VM に CPU を割り当てられるため，他プロセスを動作させることができる．また，`sched_yield()` システムコールの終了を VMM により捕捉することで，代理プロセスから代理実行の結果が返却された際，結果を重要プロセスに返却できる．

1 つの保護対象 VM 上に重要プロセスが複数動作する場合，ある重要プロセスのシステムコール情報を作成した後，図 3 (A) により保護対象 VM に制御を戻したときに，別の重要プロセスが代理実行する対象のシステムコールを発行する可能性がある．この場合，2 つ目のシステムコール情報を作成して代理実行する必要がある．しかし，複数のシステムコール情報をどのように管理するかは未検討である．また，複数の代理実行を効率的に行う方法も検討する必要がある．このため，上記は今後の課題である．

4. 評価

4.1 目的と環境

提案手法の有効性を示すために以下を評価した．

(1) 保護対象 VM の停止時間の削減効果
(2) 他プロセスの性能向上効果
(3) 重要プロセスの性能向上効果
(4) `sched_yield()` システムコールのオーバヘッド

評価モデルを**図 4** に示す．本評価では，保護対象 VM を n 台動作させ，各保護対象 VM 上で動作する重要プロセスの数を 1 とする．代理 VM 上では重要プロセスと同数，つまり，n 個の代理プロセスを走行させる．なお，重要プロセス i（$1 \leq i \leq n$）の代理実行は，代理プロセス i が行う．また，各 VM に仮想 CPU を 1 つ与え，それぞれに異なる 1 つの物理 CPU コアを固定で割り当てる．これは，複数の仮想 CPU が同じ物理 CPU コアを共有することで，各仮想 CPU の物理 CPU コアへの割り当てが不安定になり，測定結果が不安定になることを避けるためである．

表 1　評価環境
Table 1　Environment of the evaluation.

計算機 A		
CPU	Intel Core i7-2600 (3.4 GHz, 4 コア)	
メモリ	8 GB	
VMM	Xen 4.2.3	
	保護対象 VM ($\times n$)	代理 VM
仮想 CPU	1 コア	1 コア
メモリ	1 GB	$8 - n$ GB
OS	Debian 7.3 (Linux 3.2.0，64 bit)	
計算機 B		
CPU	Intel Core i7-4790 (3.6 GHz, 4 コア)	
メモリ	8 GB	
OS	Debian 9.0 (Linux 4.9.0，64 bit)	

評価環境を**表 1** に示す．計算機 A では，図 4 に示した構成で保護対象 VM と代理 VM が動作する．計算機 B では，保護対象 VM 上の重要プロセスが VM 外のプロセスと通信する際の通信相手となるプロセスが動作する．

4.2 保護対象 VM の停止時間の削減効果

提案手法による保護対象 VM の停止時間の削減効果を示すために，重要プロセスが発行した `sendto()` システムコールを 1,000 回代理実行した際の保護対象 VM の停止時間を測定した．`sendto()` の送信サイズは，1 KB，2 KB，3 KB，および 4 KB とし，保護対象 VM の数を 1，2，および 3 とした場合についてそれぞれ測定した．計算機 B では，`recvfrom()` システムコールを繰り返し発行する受信プロセスが重要プロセスと同じ数だけ動作する．また，代理プロセスがシステムコール情報の有無をポーリングすること（図 1 (5)）による保護対象 VM の停止時間への影響を取り除くために，ポーリング間隔を 0 秒とした．

測定結果を**図 5** に示す．図 5 より，以下のことがわかる．

(1) 1 KB〜4 KB のどの送信サイズにおいても，また，1 VM〜3 VM のどの VM 数においても，提案手法により保護対象 VM の停止時間を削減できている．この理由は，既存手法では VMM が代理実行の終了待ちをする（図 1 (4)）一方で，提案手法では VMM から保護対象 VM に制御を戻し（図 3 (4′)），結果返却の確認時のみ VMM が動作するためである．

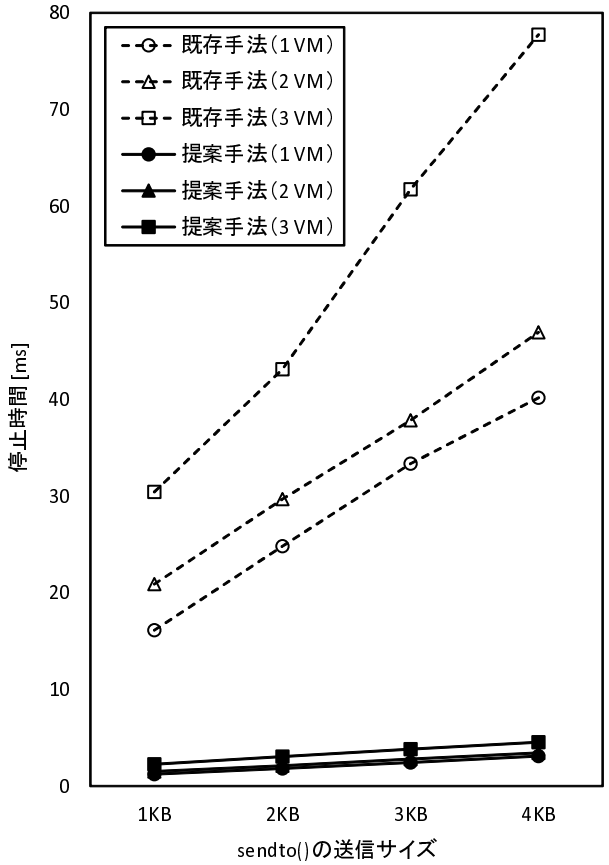

図 5 sendto() を 1,000 回代理実行した際の保護対象 VM の停止時間

Fig. 5 Downtime of the protection target VM for proxying sendto() 1,000 times.

(2) 1 VM〜3 VM のどの VM 数においても，送信サイズが大きいほど，提案手法により削減した保護対象 VM の停止時間は大きい．この理由は，提案手法により削減できる保護対象 VM の停止時間が代理プロセスの動作する時間に依存し，代理プロセスによる sendto() システムコールの実行時間が送信サイズに依存するためである．なお，削減できた保護対象 VM の停止時間の割合は，どの送信サイズにおいても 90%以上である．

(3) 既存手法と提案手法において，1 KB〜4 KB のどの送信サイズにおいても，保護対象 VM の増加に伴って，保護対象 VM の停止時間は増加する．この理由は，代理 VM 上で動作する代理プロセスの数が増加したことにより，スケジューリングによる待ち時間が発生し，代理プロセスの応答が遅延したためである．

4.3 他プロセスの性能向上効果

提案手法により保護対象 VM の停止時間を削減した場合における保護対象 VM 上の他プロセスが行う以下の処理の性能向上効果を示す．

(1) getpid() システムコールの処理性能

(2) read() システムコールの処理性能

この理由を以下に示す．完全仮想化された VM 上で read() システムコールが発行され，外部記憶装置へのアクセスが発生すると，Domain-0 と呼ばれる VM で，外部記憶装置へのアクセスをエミュレートする処理が実行される．また，既存手法と提案手法では，代理実行のオーバヘッドを小さくするために，Domain-0 を代理 VM として使用している．このため，保護対象 VM や代理 VM の動作が複雑化し，read() システムコールの処理性能の評価だけでは，提案手法による性能向上効果の分析が難しい．以上のことから，処理内容が単純で保護対象 VM 内で処理が完結する getpid() システムコールの処理性能を示した後，read() システムコールの処理性能を示す．

getpid() システムコールや read() システムコールの性能測定を行う他プロセスは，保護対象 VM(1) 上で動作し，重要プロセスが発行した sendto() システムコールが 1,000 回代理実行される間の各システムコールの処理性能（1 ms あたりの発行回数）を測定する．重要プロセスは，1 回目のシステムコールを発行する直前と 1,000 回目のシステムコールを発行した直後に，測定を行う他プロセスに対してそれぞれ測定の開始と終了に対応するシグナルを送信する．これにより，測定を行う他プロセスに測定の開始と終了を通知する．

getpid() システムコールの性能測定においては，ライブラリキャッシュの影響を避けるために，syscall 関数を用いて getpid() システムコールを発行した．また，read() システムコールの性能測定においては，ディスクキャッシュの影響を避けるために，ディスク（RAW デバイス）上のランダムな 4 KB 境界の位置から 4 KB のデータを読み込むようにした．なお，sendto() システムコールの送信サイズは，1 KB，2 KB，3 KB，および 4 KB とし，保護対象 VM の数を 1，2，および 3 とした場合についてそれぞれ測定した．計算機 B では，recvfrom() システムコールを繰り返し発行する受信プロセスが重要プロセスと同じ数だけ動作する．また，代理プロセスがシステムコール情報の有無をポーリングすること（図 1 (5)）による保護対象 VM の停止時間への影響を取り除くために，ポーリング間隔を 0 秒とした．

他プロセスが発行するシステムコールの処理性能を**図 6** に示す．図 6 (A) は，getpid() システムコールの処理性能を示し，図 6 (B) は，read() システムコールの処理性能を示す．図 6 (A) より，以下のことがわかる．

(1) 1 KB〜4 KB のどの送信サイズにおいても，また，1 VM〜3 VM のどの VM 数においても，提案手法により getpid() システムコールの処理性能を向上できている．この理由は，提案手法で代理実行中に保護対象 VM に制御を戻すことにより，他プロセスが動作する機会が与えられたためである．

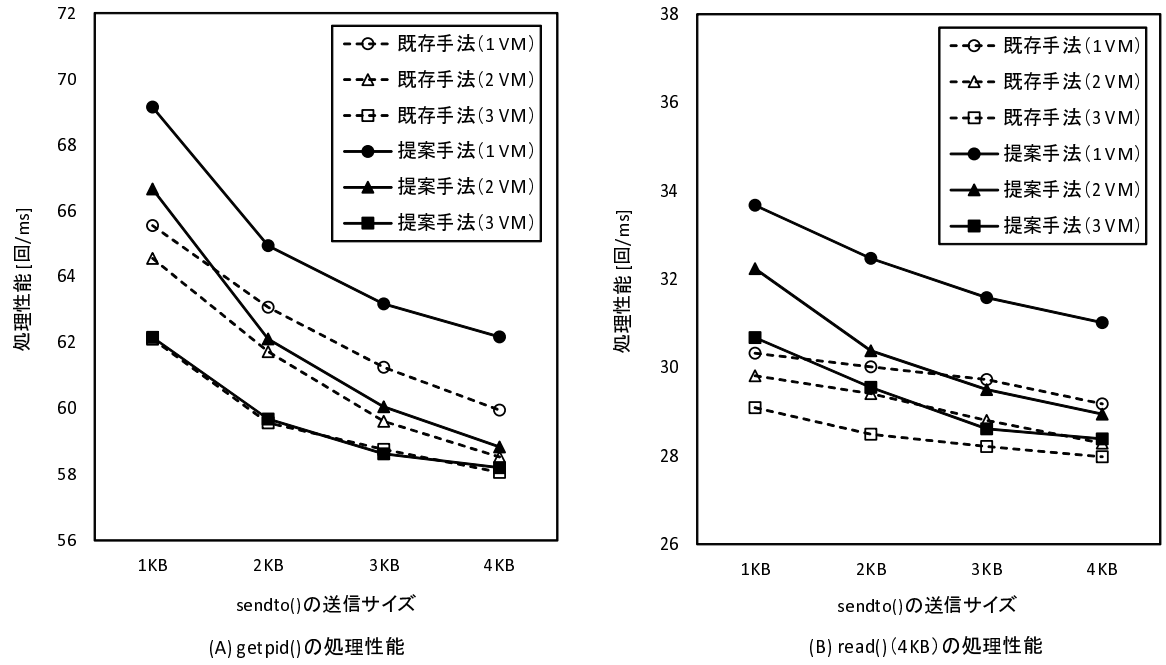

図 6　他プロセスが発行するシステムコールの処理性能
Fig. 6　Performance of system call invoked by the other process.

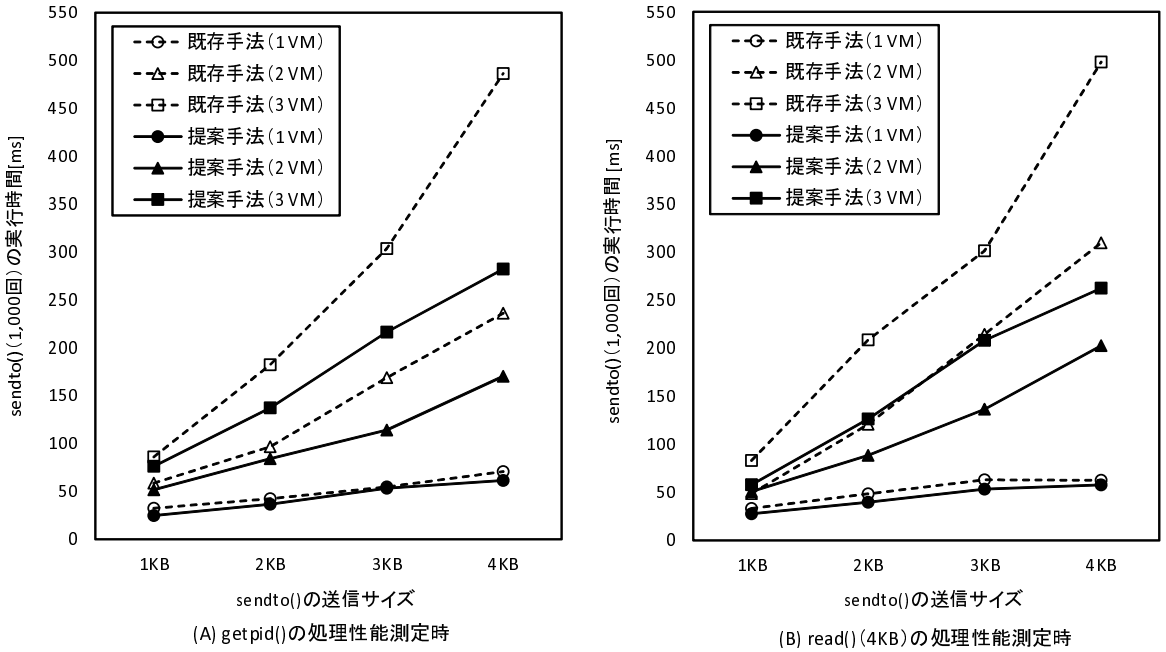

図 7　重要プロセスが発行する 1,000 回の `sendto()` の実行時間
Fig. 7　Processing time of `sendto()` 1,000 times invoked by the essential process.

(2) 1 VM〜3 VM のどの VM 数においても，送信サイズが大きくなるにつれて，`getpid()` システムコールの処理性能は低下している．この理由は，保護対象 VM の停止時間が増大したために，他プロセスが動作する時間が減少したためである．

また，図 6 (B) より，以下のことがわかる．

(3) (1) と (2) と同様のことが言える．

(4) 保護対象 VM の数が 3 つの場合において，図 6 (A) では提案手法の適用による `getpid()` システムコールの処理性能の向上がほとんどない一方で，図 6 (B) では提案手法の適用による `read()` システムコールの処理性能の向上がある．この理由は，`read()` システムコールにより外部記憶装置へのアクセスが発生した際，外部記憶装置へのアクセス中に，測定を行うプロセス以外のプロセスが動作する機会が得られるためである．このため，`getpid()` システムコールを繰り返すより

も，read() システムコールを繰り返すほうが測定を行うプロセス以外のプロセスが動作する機会が多い分，測定を行うプロセスが動作する機会が多くなるため，read() システムコールの方が，getpid() システムコールより，提案手法の適用による処理性能の向上効果が大きい．

4.4 重要プロセスの性能向上効果

保護対象 VM 上の他プロセスの処理性能が向上したことによる，重要プロセスの処理性能の向上効果を示す．**図 7** に，図 6 の測定において，重要プロセスが発行した 1,000 回の sendto() システムコールの実行時間を示す．図 7 (A) は，getpid() システムコールの処理性能の測定時の実行時間を示し，図 7 (B) は，read() システムコールの処理性能の測定時の実行時間を示す．図 7 より，以下のことがわかる．

(1) 図 7 の (A) と (B) のどちらにおいても，提案手法により 1,000 回の sendto() システムコールの実行時間は短縮できている．この理由は，提案手法により他プロセスに動作する機会を与えたことにより，他プロセスが動作した時間が増加したために，重要プロセスが動作した時間が相対的に少なくなり，重要プロセスにスケジュールされる機会が増加したためである．これにより，重要プロセスが行う 1,000 回の sendto() システムコールの実行を早く終えられるため，実行時間が短縮する．

(2) 1 VM〜3 VM のどの VM 数においても，送信サイズが大きくなるにつれて，提案手法の効果が大きく表れている．この理由は，送信サイズが大きくなるにつれて，代理プロセスによる sendto() システムコールの実行時間が大きくなり，提案手法により保護対象 VM に制御を戻す機会が増加したためである．これにより，送信サイズが大きくなるにつれて，他プロセスが動作した時間がより増加するため，重要プロセスが動作した相対的な時間がさらに少なくなる．このため，重要プロセスがスケジュールされる機会がより増加し，重要プロセスの処理性能が向上する．

4.5 sched_yield() のオーバヘッド

提案手法により，sched_yield() システムコールの終了時に，代理実行の結果返却の有無を確認する処理を追加したため，保護対象 VM 内で実行される sched_yield() システムコールにオーバヘッドが生じる．**表 2** に sched_yield() システムコールのオーバヘッドを示す．表 2 より，提案手法を適用した際，sched_yield() システムコールに 4.25 μs のオーバヘッドが生じることがわかる．オーバヘッドのほとんどは，保護対象 VM と VMM 間のモード遷移である．この理由は，sched_yield() システムコールの終了処理を捕捉した際に VMM が行う処理が，重要プロセスが否かを判定する処理，結果返却の有無を確認する処理，および結果を保護対象 VM のレジスタやメモリに格納する処理であり，他プロセスが sched_yield() システムコールを発行する際は，重要プロセスか否かを判定する処理のみ実行されるためである．

表 2 sched_yield() のオーバヘッド (μs)
Table 2 Overhead of sched_yield() (μs).

	適用前	適用後	オーバヘッド
実行時間	0.15	4.40	4.25

5. 関連研究

システムコールを代理実行する手法として，ProxyOS [8] や Shadow Context [9] がある．ProxyOS は，アプリケーションと OS のインタフェースであるシステムコールの信頼性をアプリケーション側で設定できる OS である．アプリケーションが信頼できないと設定したシステムコールは，アプリケーションが動作する VM とは別に用意した信頼できる VM 上で実行される．ProxyOS は，システムコールを代理実行するために，OS のソースコードを変更している．Shadow Context は，信頼できる VM 上で動作する ps や lsmod などの調査プログラムが発行したシステムコールを VMM により捕捉し，信頼できない VM 上の任意のプロセスにシステムコールを実行させる．これにより，信頼できない VM とは別の信頼できる VM 上で調査プログラムを実行させながら信頼できない VM を調査できるため，調査プログラムを信頼できない VM 上で実行することを回避できる．Shadow Context は，信頼できない VM を調査するために，調査プログラムが動作する OS のソースコードを変更している．これらの手法とは異なり，提案手法は，攻撃者から提案手法の存在を検知されることを避けるために，OS や重要プロセスのソースコードを変更しておらず，VMM と代理プロセスにより実現している．

OS の性能低下を回避する手法として，PicoDriver [10] がある．PicoDriver は，最小限のデバイスドライバ機能を軽量カーネルに実装し，残りのデバイスドライバ機能を Linux カーネルから利用するデバイスドライバである．これにより，軽量カーネルの性能低下を回避しつつ，軽量カーネルにデバイスドライバを適用することができる．PicoDriver は，デバイスドライバを分割することにより軽量カーネルの性能低下を回避している一方で，提案手法は，保護対象 VM に制御を戻してプロセスが動作する機会を増やすことにより，保護対象 VM 上の OS の性能低下を回避している．

VM 内のプログラムコードを利用する手法として，Virtuoso [11] がある．Virtuoso は，VM 内のプログラムの実行をトレースしてプログラムコードを収集し，VMM 内で

動作する VM 監視ツールを自動生成する．これにより，OS の内部動作に関する知識なしに VM 監視ツールを生成できる．Virtuoso は，VM 内のプログラムコードをもとに VM 監視ツールを自動生成するという形で VM 内のプログラムコードを利用する一方で，提案手法は，重要プロセスが `sched_yield()` システムコールを発行したかのように保護対象 VM を動作させるという形で VM 内のプログラムコードを利用する．

6. おわりに

システムコールの代理実行における保護対象 VM の停止を回避する手法を述べた．具体的には，システムコールを代理実行する間，あたかも重要サービスが `sched_yield()` システムコールを発行したかのように保護対象 VM を動作させることで，重要サービス以外のプロセスに動作する機会を与える．これにより，VMM が保護対象 VM の CPU を使用して動作する時間が減少し，代理実行中に保護対象 VM が停止することを回避できる．評価では，提案手法により，重要プロセスの処理を代理実行している間に VMM が保護対象 VM の CPU を占有する時間を 90 %以上削減できることを示した．また，保護対象 VM 上で動作する他プロセスの処理性能を向上できることを示し，他プロセスの処理性能の向上により，重要プロセスの処理性能も向上できることを示した．

謝辞 本研究の一部は，JSPS 科研費 18K18051 の助成を受けたものです．

参考文献

[1] Min, B. and Varadharajan, V.: A Novel Malware for Subversion of Self-protection in Anti-virus, *Softw. Pract. Exper.*, Vol. 46, No. 3, pp. 361–379 (2016).

[2] Stealth: A new Adore root kit, `http://lwn.net/Articles/75990/` (accessed 2018-1-19).

[3] Hsu, F.-H., Wu, M.-H., Tso, C.-K., Hsu, C.-H. and Chen, C.-W.: Antivirus Software Shield Against Antivirus Terminators, *IEEE Transactions on Information Forensics and Security*, Vol. 7, pp. 1439–1447 (2012).

[4] Srinivasan, D., Wang, Z., Jiang, X. and Xu, D.: Process Out-grafting: An Efficient "out-of-VM" Approach for Fine-grained Process Execution Monitoring, *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pp. 363–374 (2011).

[5] Sato, M., Taniguchi, H. and Yamauchi, T.: Design and Implementation of Hiding Method for File Manipulation of Essential Services by System Call Proxy using Virtual Machine Monitor, *International Journal of Space-Based and Situated Computing*, Vol. 9, No. 1, pp. 1–10 (2019).

[6] Okuda, Y., Sato, M. and Taniguchi, H.: Implementation and Evaluation of Communication-Hiding Method by System Call Proxy, *International Journal of Networking and Computing*, Vol. 9, No. 2, pp. 217–238 (2019).

[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, *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pp. 164–177 (2003).

[8] Ta-Min, R., Litty, L. and Lie, D.: Splitting Interfaces: Making Trust Between Applications and Operating Systems Configurable, *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, pp. 279–292 (2006).

[9] Wu, R., Chen, P., Liu, P. and Mao, B.: System Call Redirection: A Practical Approach to Meeting Real-World Virtual Machine Introspection Needs, *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 574–585 (2014).

[10] Gerofi, B., Santogidis, A., Martinet, D. and Ishikawa, Y.: PicoDriver: Fast-path Device Drivers for Multi-kernel Operating Systems, *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, pp. 2–13 (2018).

[11] Dolan-Gavitt, B., Leek, T., Zhivich, M., Giffin, J. and Lee, W.: Virtuoso: Narrowing the Semantic Gap in Virtual Machine Introspection, *2011 IEEE Symposium on Security and Privacy*, pp. 297–312 (2011).