Vol.2017-CSEC-78 No.27
Vol.2017-SPT-24 No.27
2017/7/14

プロセス管理表へのアクセス制御機能の評価

佐藤 将也[1,a)]　山内 利宏[1]　谷口 秀夫[1]

概要：セキュリティソフトウェアや証拠保全のためのソフトウェア (以降，重要サービス) は，攻撃者による計算機内の情報の取得，情報の漏洩，および被害の拡大を防止するために重要である．著者らは，重要サービスへの攻撃を回避するために，重要サービスを提供するプロセスに関する情報の参照を制限することで，重要サービスの識別を困難にする手法を提案した．また，提案手法を仮想計算機モニタにより実現した．本稿では，プロセス情報を管理する重要なプロセス管理表へのメモリアクセス制御手法について，有効性を評価した結果を報告する．評価では，攻撃への耐性について，Linux を対象としたっマルウェアを調査し，提案手法により重要プロセスの識別を回避できるか確認した．また，重要プロセスとその他のプロセスについて，提案手法による性能への影響を評価した．

1. はじめに

計算機への攻撃の防止や被害状況の把握のために，セキュリティソフトウェアや証拠保全のためのソフトウェア (以降，重要サービス) を攻撃から保護することが重要である．これらの重要サービスは，攻撃者にとっては攻撃の障害となるため，攻撃の対象になる場合がある．重要サービスが攻撃により停止または無力化されると，被害状況の把握が困難になり，攻撃や障害による被害が拡大する可能性がある．

これらの問題への対処として，重要サービスへの攻撃を防止するための手法が研究されている [1], [2], [3]．ANSS [1] は，Windows においてプロセスの終了に用いられる API を監視し，停止対象のプロセスが事前に定義したセキュリティソフトウェアであった場合は，API の実行を中断させることで，セキュリティソフトウェアを不正に停止させる攻撃を防止する．しかし，カーネルレベルでの攻撃により無効化される可能性がある．このような問題への対処として，仮想化技術を用いることで，攻撃への対策を OS から隔離して実現する方法が提案されている [2], [3]．しかし，重要サービスを提供するプログラムを改変なしには利用できない問題がある．

これらの問題への対処として，文献 [4] において，プロセス情報へのメモリアクセス制御手法を提案し，文献 [5] において，基本評価を述べた．提案手法は，プロセス管理表へのメモリアクセスを仮想計算機モニタ (Virtual Machine Monitor, 以降，VMM) により制御することで，許可したプログラム以外には偽の情報を見せる．提案手法は，アクセス制御機構を VM 外部に実現し，かつ重要サービスの改変なしに実現する．これにより，既存の重要サービスを改変なしに保護可能であり，アクセス制御機構への攻撃が困難な機構を実現した．本稿では，プロセス管理表へのメモリアクセス制御機能について，有効性の評価として攻撃への耐性と性能を評価した結果を報告する．

[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)] sato@cs.okayama-u.ac.jp

2. プロセス管理表へのアクセス制御機能

2.1 考え方

本章では，文献 [4] で提案したプロセス情報へのアクセス制御による攻撃回避手法について，特に，プロセス管理表を対象としたメモリアクセス制御機能を述べる．

提案手法では，重要プロセスのプロセス管理表の読み込みを検知し，読み込み元のアドレスに応じて書き換える．具体的には，予め許可されていないカーネルモジュールなどがプロセスの情報を読みこんだ場合には，偽の情報に置換する．これにより，攻撃者がカーネルモジュールを用いてプロセスを探索し，セキュリティソフトウェアとして動作するプロセスを強制終了させることで攻撃を検知されないようにする攻撃に対して，セキュリティソフトウェアが攻撃される可能性を低減できる．また，重要プロセス自身は，自プロセス管理表の情報を読み込めるため，提案手法の適用により重要プロセスの動作を妨げることはない．

2.2 基本方式

本稿では，重要サービスとは，計算機を攻撃者から保護

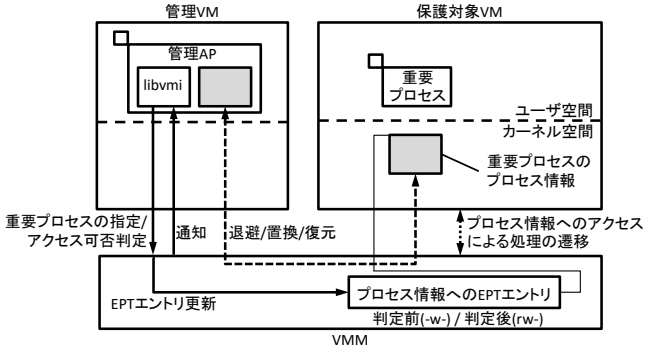

図 1　提案手法の全体像

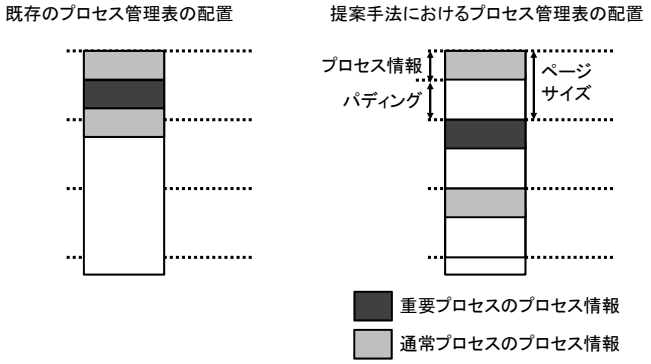

図 2　プロセス情報のメモリ上の配置

するプログラムや計算機の管理のために用いられ，定常的に走行するプログラムと定義する．また，重要サービスを提供するプロセスを重要プロセスとする．提案手法は，重要プロセス以外のプロセスやカーネルモジュールから重要プロセスの識別を困難にする手法であり，重要プロセス自体の脆弱性などを対象とした攻撃を防止するものではない．

提案手法の全体像を**図 1** に示す．提案手法では，保護対象 VM 上で動作する重要プロセスのプロセス情報へのアクセスを VMM により監視し，保護対象 VM とは異なる管理 VM 上の管理 AP により制御する．アクセスを許可する場合は，何もせず保護対象 VM に処理を戻す．アクセスを拒否する場合には，アクセス先のプロセス情報が重要プロセスのものであると識別されないように，偽の情報に置換し，保護対象 VM に処理を返却する．この際，プロセス情報が偽の情報のままでは，重要プロセスが走行する際に本来の情報を参照できないため，偽の情報へのアクセスの後に，本来のプロセス情報へ復元する．VM と VMM は隔離されており，VM 上のカーネルで攻撃者の攻撃コードが走行したとしても，VMM への攻撃は困難である．また，VMM は，VM にメモリを提供していることから，VM 内のメモリの利用状況の把握や管理が可能である．これらのことから，提案手法では，VMM を用いてプロセス情報へのアクセス制御を行う．

2.3　プロセス情報

プロセス情報とは，攻撃者がそのプロセスを特定するために利用する情報である．本稿では，特に，プロセス管理表をプロセス情報として扱う．プロセス管理表には，プロセス ID（以降，PID），プロセス名，プロセスの利用するファイル，およびプロセスが利用するメモリ領域へのポインタなど，プロセスを特定するために利用可能な情報が多く含まれている．

2.4　メモリアクセス制御

提案手法は，Extended Page Table (EPT) を用いてプロセス情報へのアクセス制御を行う．具体的には，EPT のページテーブルのエントリを操作し，読み込みを不可とする．これにより，プロセス情報の読み込みに応じて，VMM へ処理を遷移させることが可能となる．このため，アクセス制御の粒度はページ単位である．また，プロセス情報ごとにアクセス制御を行うために，プロセス情報の定義を変更し，1 つのプロセス情報が 1 ページを占有するようにした．**図 2** にプロセス情報のメモリ上の配置方法を示す．これにより，プロセス情報単位でのアクセス制御が可能になる．詳細は文献 [4] で述べた．

2.3 節で述べたように，本稿では，プロセス管理表をプロセス情報としている．ここで，プロセス管理表のサイズは，本稿で評価に用いた Linux 3.2.26 では 1,776 バイトであり，ページサイズよりも小さい．このため，1 ページごとにメモリアクセス制御を行うことで，プロセス管理表へのアクセスを制御できる．プロセス情報として扱うデータのサイズがページサイズよりも大きい場合は，サイズに応じたページ数を監視しアクセスを制御する必要がある．

図 3 に重要プロセスのプロセス情報へアクセスした際の処理の流れを示す．プロセス情報が格納されたページの EPT エントリを操作して読み込みを禁止したページへの読み込みが発生すると，EPT violation が発生し，VMM に処理が遷移する．処理の遷移後，命令ポインタをもとにアクセス元アドレスが許可した範囲に含まれるか否かを判定する．範囲に含まれていれば，カーネルスタックから戻りアドレスを取得し，すべての戻りアドレスが許可した範囲に含まれるか判定する．すべて含まれている場合は，プロセス情報の読み込みを許可する．これらのいずれの条件にも合致しなかった場合は，プロセス情報を退避し，プロセス情報を偽の情報に置換する．これらの処理が完了した後に，プロセス情報の読み込みを許可する．ただし，次回読み込み発生時に読み込みを VMM から検知できないため，シングルステップモードを設定し，ゲスト OS へ処理を返却する．

図 4 にシングルステップモードで VMM に処理が遷移した際の処理の流れを示す．シングルステップモードでゲスト OS の処理が実行されると，VMM に処理が遷移する．VMM に処理が遷移すると，重要プロセスのプロセス情報

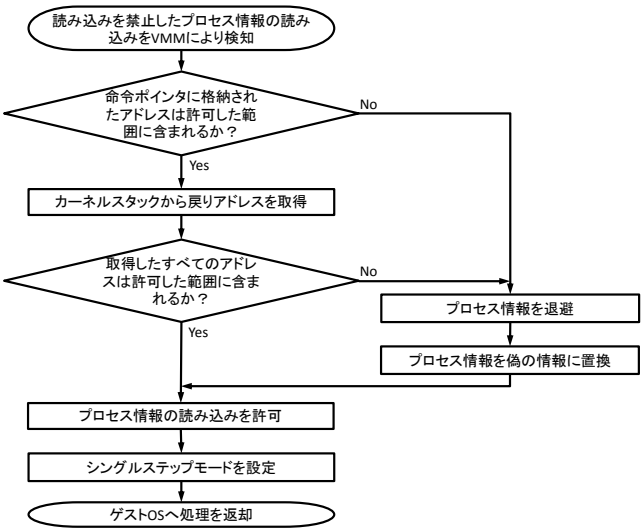

図 3 重要プロセスのプロセス情報へのアクセスが発生した際の処理

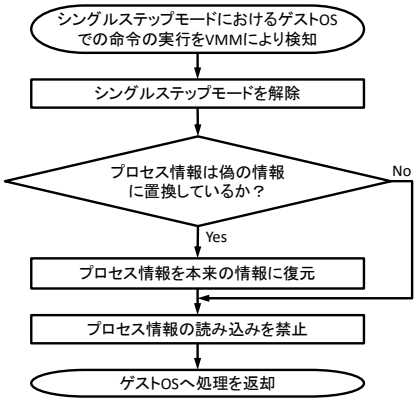

図 4 重要プロセスのプロセス情報へのアクセスが発生した際の処理

が偽の情報に置換されているかを判定し，置換されている場合は，本来の情報に復元する．これらの処理を行った後，プロセス情報の読み込みを禁止し，ゲスト OS へ処理を返却する．

3. 評価

3.1 目的と環境

文献 [5] において，プロセス特定の回避，提案手法の適用におけるコードの改変量，基本的な性能評価，およびメモリ使用量の増加について，評価結果を報告した．しかし，プロセス特定の回避については，重要プロセスの停止手段として killall コマンドを用いた場合に重要プロセスが停止しなかったことを確認したのみである．また，性能評価では，提案手法を適用した場合において，重要プロセスのプロセス管理表を読み込む処理の処理時間を測定したのみであり，応用プログラムを用いた際の性能については評価していない．また，重要プロセス数 1 の場合しか評価されていない．これらのことから，本稿では，以下の目的で評価を行った結果を述べる．

(1) 重要サービス停止への耐性
提案手法によるプロセス管理表へのアクセス制御機能が攻撃に対して有効か評価した．具体的には，実在するマルウェアによりプロセスを終了するために用いられる手段を調査し，これらの手段に対して提案手法が有効か否かを調査した．

(2) 性能評価
提案手法による保護対象 VM の性能への影響を評価した．具体的には，プロセス情報へのアクセスによる処理時間の増加量の評価と応用プログラムの性能評価を行った．応用プログラムの性能評価では，応用プログラムを重要プロセスとして指定した場合の性能低下を評価した．また，重要プロセスが存在する環境において，重要プロセス以外のプロセスの性能へ提案手法が与える影響を評価した．

表 1 評価環境

ソフトウェア	
VMM	Xen 4.2.3
OS	保護対象 VM: Debian 7.3 (Linux 3.2.0 64-bit)
	ファイル提供 VM: Debian 7.3 (Linux 3.2.65 64-bit)
ハードウェア	
CPU	Intel Core i7-2600, 3.40 GHz, 4 コア
	管理 VM: 4 仮想 CPU
	保護対象 VM: 1 仮想 CPU
メモリ	管理 VM: 15 GB
	保護対象 VM: 1 GB

表 1 に評価環境を示す．VMM には，Xen [6] を用いた．また，保護対象 VM の仮想化には，Intel VT-x と EPT を用いた．管理 AP から保護対象 VM の EPT を操作するために libvmi [7] を用いた．管理 VM の仮想 CPU は CPU コア 0 を使い，保護対象 VM の仮想 CPU は CPU コア 3 を使うように設定して評価を行った．これは，CPU コアを共有した場合に管理 VM の処理が保護対象 VM の性能に影響することを防ぐためである．なお，2.4 節で述べたように，保護対象 VM 上で動作するカーネルは，プロセス管理表がページ単位で確保されるように改変されたものを用いた．

3.2 重要サービス停止への耐性

重要サービス停止に対する提案手法の有効性を明らかにするために，マルウェアによるプロセス停止手段と停止対象を調査した．評価環境では，保護対象 VM 上の OS として Linux を用いる．このため，本評価では，Linux を対象としたマルウェア 474 検体を調査し，プロセスを終了させる機能を持つ 16 検体を用いた [8]．なお，プロセスを終了させる処理が含まれている場合でも，自分自身を終了させるためにコマンドを実行している検体は除外した．

表 2 に評価に用いたマルウェアの情報を示す．マルウェア欄には検体名，停止手段にはプロセスの停止に使用され

たコマンド名，停止対象には停止手段の引数として与えられた文字列を示している．停止対象が傍線で示されているものは，攻撃者からコマンドで指定されたプログラムを停止するものである．

主な停止手段として，kill, pkill, および killall の 3 種類のコマンドが考えられる．本調査で取得した検体では，プロセスの停止手段として用いられていたものは kill コマンドと killall コマンドであった．また，kill コマンドを用いるものの多くは自分自身や自分自身に関係するプロセスを終了させるために用いられており，特定のプロセスを終了させるような処理の多くには killall コマンドが用いられていた．

kill コマンドを用いたプロセスの停止について，停止対象が pid で指定されている場合は，pid を特定するために pidof コマンドが用いられていた．pidof コマンドは，引数で与えられた文字列をコマンド名に持つプロセスの pid を procfs から取得するコマンドであり，killall コマンドと同じであるとみなすことができる．

killall コマンドは，procfs を用いてコマンド名から pid を取得し，kill システムコールにより SIGKILL シグナルを送信している．procfs を用いてコマンド名を取得した場合，Linux カーネル内で `get_task_comm` 関数により `task_struct` 構造体からコマンド名を取得する．このため，提案手法によりプロセス管理表 (`task_struct` 構造体) へのアクセスを監視する手法は，表 2 で示すマルウェアに対して有効であるといえる．

表 2 で示す停止手段を停止対象に対して実行した場合に，提案手法により停止を回避できるかを確認した．ただし，評価で用いたマルウェアはいずれも古いため，表 1 で示す環境で動作するプログラムに読み替えて実験を行った．実験の結果，killall コマンドを実行した場合は `no process found` と表示され，pidof コマンドを実行した場合は何も表示されなかった．このことから，マルウェアで用いられるコマンドによる重要プロセスの停止を回避できたことを確認した．

表 2 評価に用いたマルウェア

項番	マルウェア名	停止手段	停止対象
1	false	kill	syslogd
2	sm4ck	killall	inetd
3	wu-ftpd-2.6.2-backdoored	kill	–
		killall	inetd
4	bl0wsshd00r67p1	kill	sshd
5	fbrk1	killall	syslogd
			inetd
6	fbrk2	killall	syslogd
7	flea	kill	syslogd
8	trNkit	killall	syslogd
9	dica	killall	atd
			portmap
			syslogd
10	hackpack	killall	sshd
11	lrk3	killall	inetd
			rpc.mountd
			portmap
			named
			inetd
12	Q-0.9	killall	httpd
13	rh61	killall	syslogd
			inetd
			sshd
			rpc.statd
			crond
14	t0rn	killall	syslogd
			inetd
			syslogd
15	pwnginx-master	killall	nginx
16	PHP-backdoors-master	killall	httpd

3.3 性能評価

3.3.1 重要プロセス数による性能の変化

提案手法は，重要プロセスのプロセス管理表へアクセスがあった際，VMM へ処理が遷移する．このため，提案手法を導入した環境では，重要プロセスのプロセス管理表へのアクセス頻度により，VM の処理性能への影響が変化する．そこで，提案手法の有無だけでなく，重要プロセス数を変化させた場合においても，プロセスの一覧を探索する処理時間を比較した．これにより，重要プロセスのプロセス管理表へのアクセス回数の違いによる VM の処理性能への影響を評価した．

評価では，ps コマンドによりプロセスの一覧を表示する際の処理時間を測定した．ps コマンドは，`/proc` 以下にあるディレクトリの一覧からプロセスの pid 一覧を取得し，各 pid 名を持つディレクトリ内から，プロセスごとの情報 (例:プロセス名) を取得し，表示する．プロセス名を取得する際，各プロセスのプロセス管理表を参照する．このため，提案手法を適用した場合は，重要プロセスのプロセス管理表を参照することで処理時間が増加する．

表 3 に ps コマンドの処理時間を示す．評価環境では，プロセス数 44 の環境で全プロセスを標示する `ps -e` コマンドを 100 回実行した際の処理時間を測定した．提案手法の測定結果は，重要プロセス数を 1 とした場合の結果である．測定結果より，提案手法の導入により ps コマンドの処理時間が約 62%増加していることが分かる．ps コマンドは，`/proc` から pid 一覧を取得し，それぞれの pid に対して `/proc/<pid>/stat` と `/proc/<pid>/status` にアクセスすることで情報を取得する．このため，1 つのプロセスの情報を表示するためには，そのプロセスのプロセス管理表に 3 回アクセスする．評価結果より，1 回 `ps -e` を実行する毎に，処理時間が約 1.3 ミリ秒増加していることから，

表 3　ps コマンドを 100 回実行した際の処理時間

	処理時間 (s)
提案手法無	0.21
提案手法有	0.34

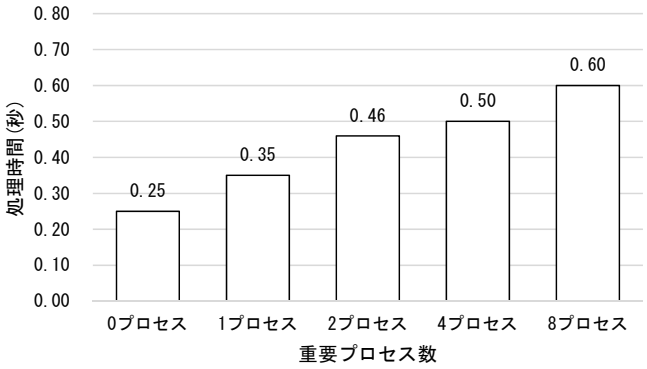

図 5　重要プロセス数による ps コマンドの処理時間の比較

提案手法を適用した場合，重要プロセスのプロセス管理表の読み込みにおけるオーバヘッドは約 0.43 ミリ秒である．

また，重要プロセス数の変化による提案手法の影響を評価するために，重要プロセスを複数走行させた際の ps コマンドの処理時間を測定した．本評価における重要プロセス数は 0, 1, 2, 4, および 8 とした．いずれの評価も，プロセス数 44 の環境で `ps -e` コマンドを 100 回実行した際の処理時間を測定した．本評価では，sleep コマンドへ十分に長い時間を引数として起動し，重要プロセスとして指定した．

図 5 に重要プロセス数による ps コマンドの処理時間の比較を示す．ps コマンドの処理時間は，重要プロセス数 0 のとき 0.25 秒であり，重要プロセス数 1, 2, 4, および 8 のときはそれぞれ，0.35 秒，0.46 秒，0.50 秒，および 0.60 秒であった．このことから，重要プロセス数が 1 増加するごとに，処理時間は約 0.10 秒増加すると推測できる．

以上の結果より，提案手法を適用した場合，重要プロセス数が 1 増加するごとに，ps コマンド 1 回の処理時間が約 1 ミリ秒増加することが分かる．

3.3.2　Web サーバの性能への影響

提案手法を適用した場合の重要プロセスの性能への影響を評価するために，Web サーバを重要プロセスとした場合の性能を評価した．評価において，Web サーバとして thttpd 2.27 を用い，ベンチマークソフトとして ApacheBench Version 2.3 を用いた．評価には 2 台の計算機を用いた．表 1 で示した計算機において，保護対象 VM 上で重要プロセスとして thttpd を実行し，1 Gbps Ethernet で接続したクライアント計算機上で ApacheBench を実行し，保護対象 VM 上の thttpd の性能を測定した．評価に用いたファイルサイズは，**表 4** で示すように 1–128 KB とした．評価では，ApacheBench を用いて並列度 1，要求回数 10,000 回の処理を各ファイルサイズ毎に 10 回ずつ行った際のスループットの平均値を用いた．

表 4　Web サーバのスループット

ファイルサイズ	スループット (MB/s)		相対性能
(KB)	Xen	提案手法	(Xen: 1)
1	3.81	1.28	0.34
2	5.97	2.42	0.41
4	11.30	4.12	0.36
8	19.42	8.63	0.44
16	32.63	15.92	0.49
32	47.40	23.52	0.50
64	64.26	28.77	0.45
128	72.51	39.51	0.54

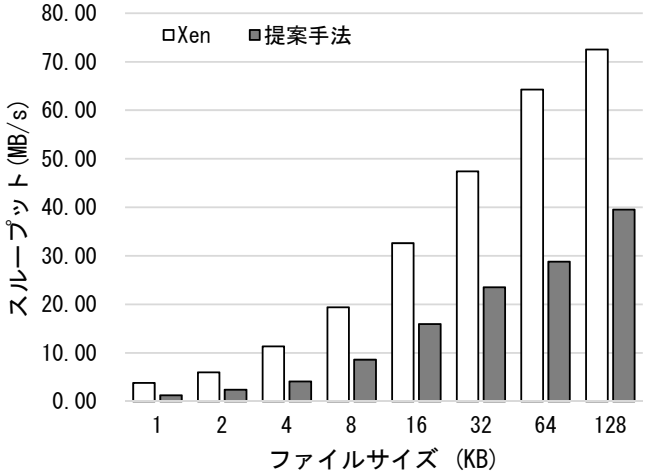

図 6　Web サーバの性能の比較

Web サーバの性能を測定した結果を表 4 に示す．また，測定結果を比較したグラフを**図 6** に示す．表 4 より，ファイルサイズが 1–8 KB の場合は，50%以下の性能である．また，ファイルサイズが 16–128 KB の場合は，約 50%程度の性能である．さらに，ファイルサイズが増加するにつれ，Xen に対する提案手法の相対性能は 1 に近づいていることが分かる．これは，提案手法によるプロセス管理表へのアクセス制御が発生する回数とその際の処理時間の増加量がファイルサイズによらず一定であるため，ファイルサイズが増加すると，ファイル転送処理に要する処理時間が全体の処理時間に占める割合が増加し，提案手法による性能低下の影響が小さくなるためであると推察する．なお，1 回のファイル要求に対して，提案手法によるアクセス制御が 10 回働いたことを確認した．また，アクセス制御の回数は，ファイルサイズによらず一定して 10 回であった．

以上の結果より，提案手法を用いた場合，Web サーバのように頻繁にプロセス管理表へアクセスする処理の性能は，最悪の場合で約 30%まで低下することが明らかになった．ただし，提案手法の適用による性能低下は，プロセス管理表へのアクセス回数に依存しており，処理負荷に関わらずプロセス管理表へのアクセス回数が一定の処理に関しては，処理負荷が増加するごとに提案手法の影響は小さく

表 5　LMbench の測定結果 (CPU とプロセスに関する性能) (μs)

	null call	null I/O	stat	open close	slct TCP	sig inst	sig hndl	fork proc	exec proc	sh proc
提案手法無	0.04	0.08	0.40	0.74	2.37	0.12	1.88	173.4	350.6	706
提案手法有	0.04	0.08	0.40	0.73	2.38	0.12	1.88	172.6	349.4	702

表 6　LMbench の測定結果 (コンテキストスイッチ) (μs)

	2p/ 0K	2p/ 16K	2p/ 64K	8p/ 16K	8p/ 64K	16p/ 16K	16p/ 64K
提案手法無	0.75	0.87	0.94	1.02	1.48	1.20	1.53
提案手法有	0.73	0.84	0.94	1.04	1.47	1.21	1.54

なることを示した．

3.3.3 LMbench

提案手法の導入による基本性能への影響を評価するために，LMbench[9] を用いた評価を行なった．提案手法は，重要プロセスのプロセス管理表へのアクセスにより VM exit が発生し，性能低下が起こる．このため，重要プロセスに関するコンテキストスイッチやスケジューリングなどの処理において，重要プロセスのプロセス管理表へのアクセスが発生すると，性能が低下することが予想される．一方で，重要プロセスに関係しない処理については，提案手法の導入による性能への影響はないと予想される．そこで，マイクロベンチマークツールである LMbench を用いて，コンテキストスイッチを含む基本性能を測定した．なお，評価では，重要サービスとして sshd を指定した．このため，LMbench に関係するプロセスのプロセス管理表は，アクセス制御の対象外である．

表 5 に CPU とプロセスの性能に関する測定結果を示す．また，**表 6** にコンテキストスイッチに関する測定結果を示す．表 6 において，Np の表記は N プロセスによる測定を示しており，MK の表記はプロセスの利用するメモリ量が M KB であることを示す．測定結果より，Xen と提案手法で性能を比較すると，処理時間の差は，表 5 においては 1%未満であり，表 6 においては 4%以内である．この結果から，提案手法の導入による影響は小さいといえる．

3.3.4 Postmark

ファイル入出力性能への影響を評価するために Postmark 1.51 を用いた評価を行った．本評価では，ブロックサイズ 512 バイトでファイルサイズ 500 から 9.77 KB の範囲のファイルに対して，I/O バッファの有効な状態で 500,000 回のトランザクションを行う処理を 10 回測定し，処理時間の平均値を求めた．本評価において，重要プロセスは sshd とし，Postmark に関するプロセスは重要プロセスとしていない．

Postmark の結果を**表 7** に示す．測定結果より，提案手法を適用した場合でも，Postmark の処理に関係するプロセスを重要プロセスとして指定しない場合は，Postmark の処理時間は増加しないことが分かる．このことから，提案手法は，重要プロセス以外のプロセスの処理性能への影響はないことが分かる．

表 7　Postmark の測定結果

	処理時間 (s)
提案手法無	12.7
提案手法有	12.7

表 8　カーネル make の処理時間

	処理時間 (s)
提案手法無	89.62
提案手法有	93.94

3.3.5 カーネル make

カーネル make の評価では，Linux 3.2.65 のソースコードを用い，不要なモジュールを構築しないよう `make allnoconfig` を実行した後，`make -j1` を実行した際の処理時間を測定した．評価では，9 回の測定の平均値を用いた．本評価においては，重要プロセスは sshd とし，カーネル make に関するプロセスは重要プロセスとしていない．

表 8 にカーネル make の処理時間を示す．表 8 より，提案手法の適用による性能低下は十分小さいことがわかる．3.3.4 項における Postmark に対して，カーネル make は，提案手法を適用した場合は約 5%処理時間が増加している．これは，Postmark と比べてカーネル make の処理時間が長く，また，起動するプロセス数も多いため，スケジューラがプロセス管理表へアクセスする回数が多いためであると推察する．

4. おわりに

本稿では，プロセス管理表へのメモリアクセス制御機能の評価について，攻撃への耐性と性能について述べた．攻撃への耐性について，提案手法の対象である Linux において重要プロセスの停止を行うマルウェアを調査し，提案手法により攻撃を回避できることを述べた．性能評価について，プロセス管理表にアクセスする処理である ps コマンドの処理時間を測定し，1 回のプロセス管理表へのアクセスで約 0.43 ミリ秒のオーバヘッドが発生することを示した．また，提案手法の適用が重要プロセスの性能へ与える影響を評価するために，Web サーバの性能を ApacheBench により評価した．評価結果より，Web サーバのスループットが最大で 70%低下することを示した．さらに，提案手法を適用した場合における重要プロセス以外のプロセスへの性能を測定し，影響は十分小さいことを示した．

謝辞　本研究の一部は JSPS 科研費 16K16067, 16H02829 の助成を受けたものです．

参考文献

[1] Hsu, F.-H., Wu, M.-H., Tso, C.-K., Hsu, C.-H. and Chen, C.-W.: Antivirus Software Shield Against Antivirus Terminators, *IEEE Trans. Inf. Forensic Secur.*, Vol. 7, No. 5, pp. 1439–1447 (2012).

[2] Jiang, X., Wang, X. and Xu, D.: Stealthy Malware Detection and Monitoring Through VMM-based "Out-of-the-box" Semantic View Reconstruction, *ACM Trans. Inf. Syst. Secur.*, Vol. 13, No. 2, pp. 12:1–12:28 (2010).

[3] Riley, R., Jiang, X. and Xu, D.: Guest-Transparent Prevention of Kernel Rootkits with VMM-Based Memory Shadowing, *Proc. 11th International Symposium on Recent Advances in Intrusion Detection*, pp. 1–20 (2008).

[4] 佐藤将也，山内利宏，谷口秀夫：プロセス情報不可視化のための仮想計算機モニタによるメモリアクセス制御，情報処理学会シンポジウムシリーズコンピュータセキュリティシンポジウム 2015 (CSS2015) 論文集，Vol. 2015, No. 3, pp. 855–860 (2015).

[5] 佐藤将也，山内利宏，谷口秀夫：プロセス情報不可視化のための仮想計算機モニタによるメモリアクセス制御機能の評価，情報処理学会研究報告，Vol. 2016–CSEC–74, No. 25, pp. 1–7 (2016).

[6] 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, *SIGOPS Oper. Syst. Rev.*, Vol. 37, No. 5, pp. 164–177 (2003).

[7] LibVMI Project: LibVMI, LibVMI Project (online), available from ⟨http://libvmi.com/⟩ (accessed 2017-06-01).

[8] Packet Storm: Files – Packet Storm, Packet Storm (online), available from ⟨https://packetstormsecurity.com/UNIX/penetration/rootkits/⟩ (accessed 2017-06-06).

[9] McVoy, L. W., Staelin, C. et al.: lmbench: Portable Tools for Performance Analysis., *USENIX annual technical conference*, pp. 279–294 (1996).