マルチコアで動作するVMに対応したKVM上の機密情報の拡散追跡機能の評価

大谷 航平[1]　山内 利宏[2,a)]　森山 英明[3]　佐藤 将也[4]　谷口 秀夫[2]

概要：近年，機密情報が漏えいする事例が増加している．この問題に対処するため，仮想計算機モニタ（Virtual Machine Monitor，以降，VMM）を利用した機密情報の拡散追跡機能（以降，VMM における拡散追跡機能）を提案した．しかし，既存の VMM における拡散追跡機能は，複数の仮想計算機（Virtual Machine，以降，VM）を同時に監視対象とした場合に必要となる排他制御，および監視対象 VM に複数の vCPU を割り当てた場合の対処について十分に検討されていない．そこで，本稿では，VMM における機密情報の拡散追跡機能について，必要な排他制御，および複数の vCPU を割り当てた VM を監視対象とした場合の対処方法について述べる．さらに，複数の vCPU を割り当てた VM を監視対象とした場合における機密情報の拡散追跡機能の追跡可能性と，機密情報の拡散追跡機能によるオーバヘッドについて評価した結果を述べる．

キーワード：情報漏えい防止，仮想化，オペレーティングシステム

Evaluation of Function for Tracing Diffusion of Classified Information to Support VMs Running on Multiple Cores on KVM

Kohei Otani[1]　Toshihiro Yamauchi[2,a)]　Hideaki Moriyama[3]　Masaya Sato[4]　Hideo Taniguchi[2]

Abstract: Recently, the cases of classified information leakage has been increasing. To solve this problem, a function to trace the diffusion of classified information using a Virtual Machine Monitor (VMM) was proposed. However, the proposed function is not considered in terms of the exclusion control that is necessary when multiple virtual machines (VMs) are monitored. In addition, the function is not examined in terms of the case where the monitoring target is a VM to which multiple virtual CPUs (vCPU) are allocated. Therefore, in this paper, we describe the exclusion control for the diffusion tracking function of classified information in VMMs. We also show how to deal with the case where a VM with multiple vCPUs is the target of monitoring. Furthermore, we report the evaluation results of the traceability of the improved proposed method and its overhead for classified information when a VM with multiple vCPUs is monitored.

[1] 岡山大学 大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University
[2] 岡山大学 学術研究院 自然科学学域
Graduate School of Natural Science and Technology, Okayama University
[3] 有明工業高等専門学校 創造工学科
Department of Creative Engineering, National Institute of Technology, Ariake College
[4] 岡山県立大学 情報工学部
Faculty of Computer Science and Systems Engineering, Okayama Prefectural University
[a)] yamauchi@cs.okayama-u.ac.jp

1. はじめに

計算機とインターネットの普及により，計算機上で機密情報を扱う機会が増加するとともに，機密情報が可搬記憶媒体やネットワークを経由し，漏えいする事例が増加している．機密情報の漏えいは，その一部が外部に漏えいした場合であっても，企業や個人にとって大きな損失となる．このため，情報漏えいの防止は重要な課題になっている．文献 [1] に示された個人情報漏えいに関するインシデントの分析結果によると，管理ミスと誤操作が個人情報漏えい

© 2021 Information Processing Society of Japan

の原因の 36.8%を占めている．この問題を解決するためには，計算機の利用者が，計算機のどこに機密情報が存在しているかを把握することが重要である．

そこで，機密情報の漏えいを未然に防ぐ手法として，機密情報が拡散される契機となるシステムコールの発行に着目し，計算機内で機密情報が拡散される状況を追跡し，機密情報を有する資源を把握する機能（以降，機密情報の拡散追跡機能）が実現されている [2]．機密情報の拡散追跡機能は，機密情報の漏えいを計算機の利用者に通知するため，利用者は，計算機外部への機密情報の書き出しを制御できる．

一方で，機密情報の拡散追跡機能は，オペレーティングシステム（以降，OS）内部に実現されているため，機密情報を窃取しようとする攻撃者や悪意のある計算機利用者からの攻撃を受けて無効化される可能性がある．また，導入の際に導入対象の OS のソースコードを修正しなければならないため，導入環境が限定される問題がある．

この問題に対処するため，仮想計算機モニタ（Virtual Machine Monitor，以降，VMM）における機密情報の拡散追跡機能が提案されている [3], [4]．VMM における拡散追跡機能は，VMM の 1 つである Kernel-based Virtual Machine（以降，KVM）内に実現されており，仮想化の方式には完全仮想化を用いている．完全仮想化環境において，仮想計算機（Virtual Machine，以降，VM）から VMM の機能を能動的に呼び出すことはできない．このため，VMM 内に機密情報の拡散追跡機能を実現した場合は，OS 内部に機密情報の拡散追跡機能を実現した場合と比較して，機密情報を窃取しようとする攻撃者や悪意のある計算機の利用者からの攻撃が困難となるため，より堅牢な機能として実現可能である．

また，VMM における拡散追跡機能は，ゲスト OS のソースコードを修正することなく，VMM のソースコードを改変することで，ゲスト OS に対して機密情報の拡散追跡機能と同等の機能を提供している．このため，VMM 内に機密情報の拡散追跡機能を実現した場合は，OS 内部に機密情報の拡散追跡機能を実現した場合と比較して，より多くの環境に導入できるという利点がある．また，VMM における拡散追跡機能を拡張して，複数の VM を監視対象とした場合において，機密情報の拡散を追跡する機能が提案されている [5].

既存の VMM における拡散追跡機能は，1 つの物理コアを，監視対象の VM 群に 1 つだけ割り当てた場合に対応している．一方で，複数の VM を監視対象とした際，各 VM における vCPU が異なる物理コアに割り当てられた場合の動作に対応していない．また，複数の vCPU を割り当てた VM を監視対象とした際，各 vCPU が異なる物理コアに割り当てられた場合の動作に対応していない．

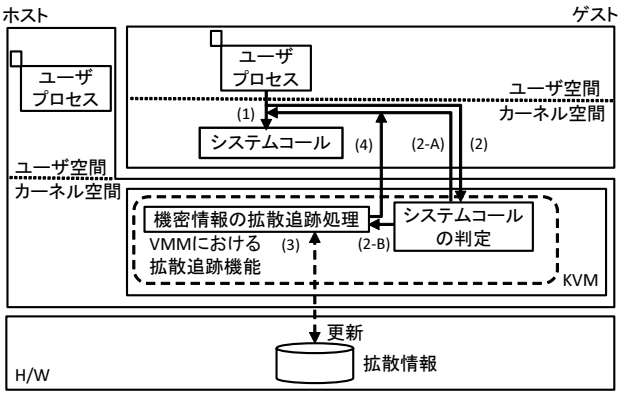

図 1　VMM における拡散追跡機能の全体図

本稿では，VMM における拡散追跡機能のマルチコア環境への対応方法を述べる．また，複数の vCPU が割り当てられた VM への対応方法を述べる．さらに，複数の vCPU が割り当てられた VM を監視対象とし，VMM における拡散追跡機能を動作させた場合のオーバヘッドを評価し，性能への影響について考察した結果を報告する．

2. VMM における拡散追跡機能

2.1 機密情報の拡散経路

文献 [3], [4], [5] で実現されている VMM における拡散追跡機能は，機密情報を有する可能性のあるファイル，プロセスを拡散情報として管理している．以降では，これらのファイルとプロセスをそれぞれ管理対象ファイルと管理対象プロセスと呼ぶ．機密情報の拡散は，プロセスがファイル形式で存在する機密情報を開いてその内容を読み込み，さらに他のプロセスやファイルなどにその内容を伝えることにより発生する．以下にプロセスが情報を拡散する 3 つの処理を示す．

(1) ファイル操作
(2) 子プロセス生成
(3) プロセス間通信

これらの処理を監視し，機密情報の拡散を追跡している．

2.2 基本機構

VMM における拡散追跡機能は，機密情報の拡散追跡機能と同等の追跡機構を VMM により実現する．具体的には，VMM において，ゲスト OS 上に存在する管理対象ファイルと管理対象プロセスを管理し，ゲスト OS 上で実行される 2.1 節で述べた 3 つの処理を監視することで，機密情報の拡散を追跡する．

VMM における拡散追跡機能の全体図を**図 1** に示し，以下で処理の流れを説明する．

(1) ゲスト OS 上でユーザプロセスがシステムコールを発行
(2) VMM でシステムコールの発行を検知し，発行された

システムコールを判定後，以下の処理を実行

(A) 機密情報の拡散に関係しないシステムコールの場合，制御をゲスト OS へ戻し，システムコールの処理を続行

(B) 機密情報の拡散に関係するシステムコールの場合，機密情報の拡散追跡に必要な情報を取得

(3) (2-B) で取得した情報をもとに機密情報の拡散を追跡し，拡散情報を更新

(4) 制御をゲスト OS へ戻し，システムコール処理を続行

上記の処理により，VMM における拡散追跡機能は，ゲスト OS のソースコードを改変することなく，機密情報の拡散を追跡することができる．VMM における拡散追跡機能により，計算機の利用者は，機密情報の所在を把握し，拡散を検知することができる．

2.3 既存機能における課題

既存の VMM における拡散追跡機能 [5] には，1 つの物理コアを，監視対象の VM 群に 1 つだけ割り当てた場合に対応している．複数の VM を監視対象とした際，各 VM における vCPU が異なる物理コアに割り当てられた場合の動作，および複数の vCPU を割り当てた VM を監視対象とした際，各 vCPU が異なる物理コアに割り当てられた場合の動作に対応させるためには，既存の VMM における拡散追跡機能には，以下の課題がある．

（課題 1） 1 台の監視対象 VM につき 1 つの vCPU のみにしかハードウェアブレークポイントを設定できない

既存の VMM における拡散追跡機能は，監視対象 VM のデバッグレジスタを用いて，VM で発行されたシステムコールをフックしている．既存の VMM における拡散追跡機能では，1 台の VM あたりに 1 つの vCPU にハードウェアブレークポイントを設定する．このため，複数の vCPU を割り当てた VM を監視対象とした場合，ハードウェアブレークポイントを設定していない vCPU 上で発行されたシステムコールを検知できない．

（課題 2） 複数の VM を監視対象とした際，それぞれの vCPU を異なる物理コアに割り当てた場合の動作に対応していない

既存の VMM における拡散追跡機能は，監視対象 VM 管理表を用いて監視対象 VM を判別している．複数の VM を監視対象とし，それぞれの VM の vCPU を異なる物理コアに割り当てた場合，監視対象 VM 管理表に必要な排他制御がされていないため，複数の物理コアが同時に監視対象 VM 管理表を更新することで，監視対象 VM 管理表が正しく更新されない可能性がある．

（課題 3） 複数の vCPU を割り当てた VM を監視対象とした際，それぞれの vCPU を異なる物理コアに割り当てた場合の動作に対応していない

既存の VMM における拡散追跡機能は，機密情報が拡散する可能性を有するファイル，プロセス，およびソケットの情報を拡散情報として管理している．複数の vCPU を割り当てた VM を監視対象とし，それぞれの vCPU を異なる物理コアに割り当てた場合，複数の物理コアが同時に拡散情報管理表を参照・更新することで，管理している情報の参照・更新が正しく行われない可能性がある．

3. マルチコアで動作する VM に対応した VMM における拡散追跡機能

3.1 対処

2.3 節で述べた課題を考慮し，マルチコアで動作する VM に対応した VMM における拡散追跡機能の実現する上での対処を以下に示す．

（対処 1） ハードウェアブレークポイントの設定

複数の vCPU を割り当てた VM を監視対象とした場合に，監視対象 VM で発行したすべてのシステムコールを検知する必要がある．このため，すべての vCPU にハードウェアブレークポイントを設置する必要がある．

（対処 2） 監視対象 VM 管理表の排他制御

監視対象 VM 管理表を参照・更新する処理を排他制御する．これにより，複数の VM を監視対象とした際，それぞれの VM における vCPU が異なる物理コアに割り当てられた場合の監視対象 VM 管理表の整合性を保証する．

（対処 3） 拡散情報管理表の排他制御

拡散情報管理表を参照・更新する処理を排他制御する．これにより，複数の vCPU を割り当てた VM を監視対象とした際，それぞれの vCPU が異なる物理コアに割り当てられた場合の拡散情報管理表の整合性を保証する．

3.2 対処内容

3.2.1 ハードウェアブレークポイントの設定

VMM における拡散追跡機能の起動処理について，すべての vCPU にハードウェアブレークポイントを設置するよう処理を変更することで 2.3 節の（課題 1）に対処した．VMexit が起きた vCPU にハードウェアブレークポイントが設置されていない場合に監視対象 VM 管理表の更新処理を実行し，ハードウェアブレークポイントを設置するよう処理を変更した．これにより，複数の vCPU を割り当てた VM を監視対象とした場合に，監視対象 VM で発行したすべてのシステムコールを検知できる．

3.2.2 監視対象 VM 管理表の排他制御

既存の VMM における拡散追跡機能では，監視対象 VM 管理表と拡散情報管理表（以降，各管理表）を参照・更新する際，複数のスレッドが同時に同一の管理表にアクセスした場合に各管理表の参照・更新が正しく行われない可能性がある．このため，排他制御する必要のある区間（以降，

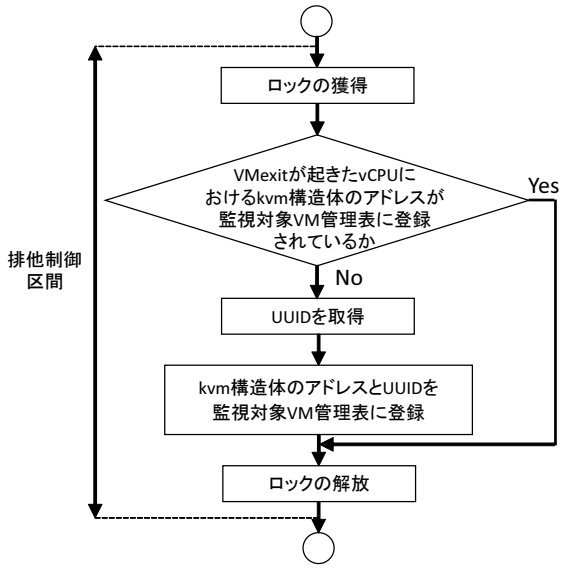

図 2 監視対象 VM 管理表の更新処理の排他制御区間

クリティカルセクション）に対して，同時に 1 つのスレッドのみがアクセスできる排他制御方式を検討する必要がある．

この条件を満たす排他制御方式の一つとして，Linux カーネルに実装されているロック機構である spinlock がある．spinlock は，クリティカルセクションに入る際に，ロックが獲得されていない場合はロックを獲得して処理を継続し，既にロックが獲得されている場合は，ロックが解放されるまでループを繰り返すロック機構である．ループを繰り返す処理は，ロックの状態を検査するために CPU 時間を使う処理として実装されている．このため，spinlock は，処理時間の少ない処理を排他制御するのに適している．VMM における拡散追跡機能において，排他制御が必要な区間の処理時間は最大で約 200μs である．これは，監視対象 VM 管理表を更新する処理の処理時間であり，VMM における拡散追跡機能の起動時に，監視対象 VM の数だけ実行される．また，拡散情報管理表を参照・更新する処理は，追跡対象のシステムコールに応じた処理中に実行され，処理時間は，約 0.1μs である．これらは，Linux カーネルにおいて，プロセスに与えられるタイムスライスの値である 20ms と比較して十分に小さい．このことから，他のロック機構より排他制御による影響が小さいと推察する．このため，本研究では，spinlock を用いて，各管理表を参照・更新する処理を排他制御した．

監視対象 VM 管理表を更新する処理は，VMM における拡散追跡機能を有効にした際に行われる．監視対象 VM 管理表を排他制御する区間を**図 2** に示す．監視対象 VM 管理表は，各 VM の一意の識別子である kvm 構造体のアドレスと UUID を対応づけて登録することで VM を判別している．このため，VMexit が起きた vCPU における kvm 構造体のアドレスが監視対象 VM 管理表に登録されているか否かを判定する処理から，kvm 構造体のアドレスと UUID を監視対象 VM 管理表に登録する処理までを排他制御区間とする．この排他制御区間の前後でロックの獲得と解放を行うことで，監視対象 VM 管理表を更新する処理を排他制御した．

表 1 追跡対象のシステムコールと参照・更新される拡散情報管理表

	参照・更新される管理表
read（ファイル操作）	管理対象プロセス管理表 管理対象ファイル管理表
write（ファイル操作）	管理対象プロセス管理表 管理対象ファイル管理表
recvfrom (UNIX ドメインソケット通信)	管理対象プロセス管理表 管理対象ソケット管理表
recvfrom (INET ドメインソケット通信)	管理対象プロセス管理表 管理対象ポート番号管理表
sendto (UNIX ドメインソケット通信)	管理対象プロセス管理表 管理対象ソケット管理表
sendto (INET ドメインソケット通信)	管理対象プロセス管理表 管理対象ポート番号管理表
clone	管理対象プロセス管理表

3.2.3 拡散情報管理表の排他制御

拡散情報管理表を参照・更新する処理の排他制御をする際，spinlock を用いて排他制御した．これは，監視対象 VM 管理表を更新する処理を排他制御する場合と同様である．

拡散情報管理表を参照・更新する処理は，追跡対象のシステムコールをフックし，各システムコールに対応した処理をする際に行われる．追跡対象のシステムコールと参照・更新される管理表の対応を**表 1** に示す．また，例として，read システムコールをフックした際の処理を**図 3** に示す．表 1 に示したシステムコールは，既存の VMM における拡散追跡機能で追跡対象としているすべてのシステムコールである．これらのシステムコールをフックした際，参照・更新する管理表を排他制御した．複数の管理表のロックを獲得する際は，デッドロックを防ぐために，管理対象プロセス管理表，管理対象ファイル管理表，管理対象ソケット管理表，管理対象ポート番号管理表の順でロックを獲得する．

3.3 期待される効果

2.3 節で述べた制限を解決することにより，以下の効果が期待できる．

（効果 1） 複数の VM を監視対象とした際，各 VM における vCPU が異なる物理コアに割り当てられた場合の動作に対応

（効果 2） 複数の vCPU を割り当てた VM を監視対象とした際，各 vCPU が異なる物理コアに割り当てられた場合の動作に対応

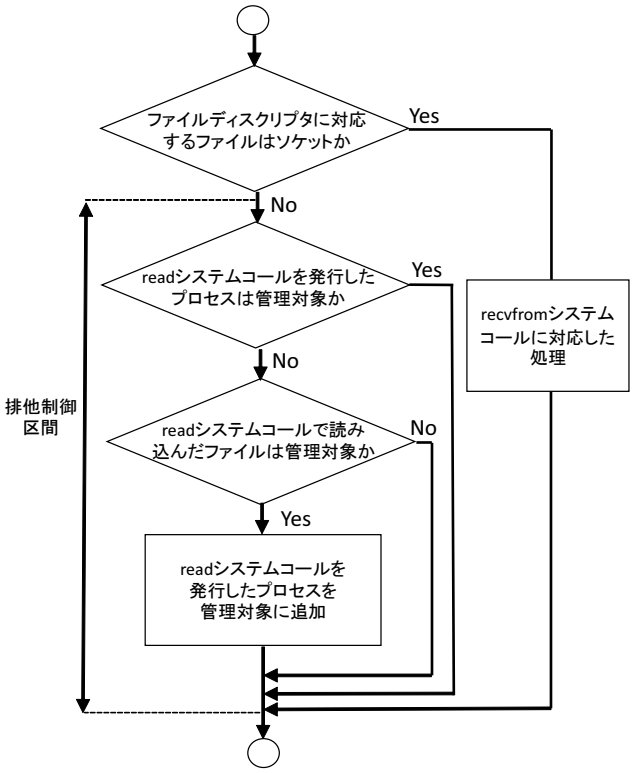

図 3 read システムコールに対応した処理

表 2 評価環境

CPU		Intel(R) Xeon E5-2609 V4（8 コア）
OS	ゲスト	Fedora 18 (Linux 3.6.10, 64 bit)
	ホスト	Fedora 18 (Linux 3.6.10, 64 bit)
メモリ	ゲスト	1 GB
	ホスト	64 GB
HDD		WDC WD1004FBYZ-0 (1 TB, 7200 rpm)
VMM		kvm-kmod-3.9

4. 評価

4.1 評価項目

複数の vCPU が割り当てられた VM を監視対象とした場合の機密情報の拡散追跡機能の追跡可能性と，機密情報の拡散追跡機能によるオーバヘッドを明確にするために，**表 2** に示す環境において，以下の項目を評価した．

（1） 機密情報の拡散追跡機能の追跡可能性

監視対象 VM に割り当てたすべての vCPU にハードウェアブレークポイントを設定できることを示すために，複数の vCPU が割り当てられた VM を監視対象とした場合において，機密情報の拡散・漏えいを検知できるか否かを示す．

（2） オーバヘッド

VMM における拡散追跡機能は，システムコールの呼び出し時の処理に機密情報の拡散追跡処理を挿入することで実現する．このため，拡散情報管理表を参照・更新する処理の排他制御により，ゲスト OS におけるシステムコールの処理時間にオーバヘッドが生じる可能性がある．VMM における拡散追跡機能を導入した際のオーバヘッドを測定することで，VMM における拡散追跡機能の性能への影響を示す．

4.2 追跡可能性

4.2.1 評価方法

4 個の vCPU を割り当てた VM を監視対象とした場合に，以下の 2 つの事例を用いて機密情報の拡散・漏えいの追跡可能性を検証した．

（事例 1） ファイル操作による機密情報の拡散

`cp` コマンドを用いて，管理対象ファイルの内容を別のファイルに書き出す．

（事例 2） 監視対象 VM 外部への機密情報の送信

`scp` コマンドを用いて，管理対象ファイルを監視対象 VM 外部であるホスト OS へ送信する．

なお，このとき taskset コマンドを用いて，監視対象 VM の各 vCPU 上でそれぞれのコマンドを実行した．

4.2.2 評価結果

`cp` コマンドを用いて管理対象ファイルを操作した際のカーネルログを**図 4** に示す．図 4 の 7 行目，14 行目，22 行目，および 29 行目では，各 vCPU 上で発行された write システムコールをフックし，機密情報の拡散を検知したことと，管理対象ファイル管理表に追加されたファイルのパス名が示されている．`cp` コマンドを用いて管理対象ファイルを操作した場合，プロセスが管理対象ファイルを読み込み，別ファイルに書き出すことによって，機密情報が拡散する．（事例 1）を行った際，VMM における拡散追跡機能は，管理対象ファイルを読み込む read システムコールをフックする．その後，読み込んだ管理対象ファイルの内容を他のファイルに書き出す write システムコールをフックした際に，機密情報の拡散を検知した．

また，`scp` コマンドを用いて管理対象ファイルを監視対象 VM の外部に送信した際のカーネルログを**図 5** に示す．図 5 の 6 行目，12 行目，18 行目，および 24 行目では，各 vCPU 上で発行された write システムコールをフックし，機密情報の漏えいを検知したことが示されている．`scp` コマンドを用いて管理対象ファイルを監視対象 VM の外部に送信した場合，監視対象プロセスが管理対象ファイルの内容を監視対象 VM の外部へ送信することによって，機密情報が漏えいする．（事例 2）を行った際，管理対象ファイルを計算機の外部に送信する write システムコールをフックした際に，機密情報の漏えいを検知した．

4.3 オーバヘッド

4.3.1 評価方法

評価項目を以下に示す．

```
1 Aug 18 16:19:34 localhost kernel: [17591.465190] write() exit
2 Aug 18 16:19:34 localhost kernel: [17591.465191] write_fd: 4
3 Aug 18 16:19:34 localhost kernel: [17591.465192] vcpu_id:3
4 Aug 18 16:19:34 localhost kernel: [17591.465195] write_pid:921
5 Aug 18 16:19:34 localhost kernel: [17591.465197] write_ino:130650
6 Aug 18 16:19:34 localhost kernel: [17591.465199] ---
7 Aug 18 16:19:34 localhost kernel: [17591.465206] sensitive data is diffused
to "home/otani/cp3" (inode number: 130650, vcpu_id: 3) by "cp" (pid: 921)
                                        ⋮
8  Aug 18 16:19:34 localhost kernel: [17591.465207] ---
9  Aug 18 16:19:34 localhost kernel: [17591.465210] write() exit
10 Aug 18 16:19:34 localhost kernel: [17591.465213] write_fd: 4
11 Aug 18 16:19:34 localhost kernel: [17591.465221] vcpu_id:1
12 Aug 18 16:19:34 localhost kernel: [17591.465221] write_pid:919
13 Aug 18 16:19:34 localhost kernel: [17591.465224] write_ino:387850
14 Aug 18 16:19:34 localhost kernel: [17591.465250] sensitive data is diffused
to "home/otani/cp1/secret.txt" (inode number: 387850, vcpu_id: 1) by "cp"
(pid: 919)
                                        ⋮
15 Aug 18 16:19:34 localhost kernel: [17591.465286] ---
16 Aug 18 16:19:34 localhost kernel: [17591.465286] write() exit
17 Aug 18 16:19:34 localhost kernel: [17591.465288] write_fd: 4
18 Aug 18 16:19:34 localhost kernel: [17591.465289] vm_index:0
19 Aug 18 16:19:34 localhost kernel: [17591.465291] vcpu_id:2
20 Aug 18 16:19:34 localhost kernel: [17591.465292] write_pid:920
21 Aug 18 16:19:34 localhost kernel: [17591.465293] write_ino:387964
22 Aug 18 16:19:34 localhost kernel: [17591.465303] sensitive data is diffused
to "home/otani/cp2/secret.txt" (inode number: 387964, vcpu_id: 2) by "cp"
(pid: 920)
                                        ⋮
23 Aug 18 16:19:34 localhost kernel: [17591.465338] ---
24 Aug 18 16:19:34 localhost kernel: [17591.465339] write() exit
25 Aug 18 16:19:34 localhost kernel: [17591.465340] write_fd: 4
26 Aug 18 16:19:34 localhost kernel: [17591.465340] vcpu_id:0
27 Aug 18 16:19:34 localhost kernel: [17591.465344] write_pid:918
28 Aug 18 16:19:34 localhost kernel: [17591.465345] write_ino:209246
29 Aug 18 16:19:34 localhost kernel: [17591.465355] sensitive data is diffused
to "home/otani/cp0/secret.txt" (inode number: 209246, vcpu_id: 0) by "cp"
(pid: 918)
```

図 4　cp コマンドを実行した際のカーネルログ

```
1 Aug 18 17:07:59 localhost kernel: [20492.779844] write() exit
2 Aug 18 17:07:59 localhost kernel: [20492.779845] write_fd: 1
3 Aug 18 17:07:59 localhost kernel: [20492.779846] vcpu_id:0
4 Aug 18 17:07:59 localhost kernel: [20492.779848] write_pid:917
5 Aug 18 17:07:59 localhost kernel: [20492.779849] write_ino:1043
6 Aug 18 17:07:59 localhost kernel: [20492.779856] sensitive data is diffused
to "tty1" (inode number: 1043, vcpu_id: 0) by "scp" (pid: 917)
                                        ⋮
7 Aug 18 17:08:12 localhost kernel: [20506.008870] write() exit
8 Aug 18 17:08:12 localhost kernel: [20506.008871] write_fd: 1
9 Aug 18 17:08:12 localhost kernel: [20506.008872] vcpu_id:1
10 Aug 18 17:08:12 localhost kernel: [20506.008875] write_pid:920
11 Aug 18 17:08:12 localhost kernel: [20506.008876] write_ino:1043
12 Aug 18 17:08:12 localhost kernel: [20506.008882] sensitive data is diffused
to "tty1" (inode number: 1043, vcpu_id: 1) by "scp" (pid: 920)
                                        ⋮
13 Aug 18 17:08:25 localhost kernel: [20519.155397] write() exit
14 Aug 18 17:08:25 localhost kernel: [20519.155398] write_fd: 1
15 Aug 18 17:08:25 localhost kernel: [20519.155399] vcpu_id:2
16 Aug 18 17:08:25 localhost kernel: [20519.155402] write_pid:923
17 Aug 18 17:08:25 localhost kernel: [20519.155403] write_ino:1043
18 Aug 18 17:08:25 localhost kernel: [20519.155409] sensitive data is diffused
to "tty1" (inode number: 1043, vcpu_id: 2) by "scp" (pid: 923)
                                        ⋮
19 Aug 18 17:08:38 localhost kernel: [20532.406067] write() exit
20 Aug 18 17:08:38 localhost kernel: [20532.406068] write_fd: 1
21 Aug 18 17:08:38 localhost kernel: [20532.406068] vcpu_id:3
22 Aug 18 17:08:38 localhost kernel: [20532.406071] write_pid:926
23 Aug 18 17:08:38 localhost kernel: [20532.406072] write_ino:1043
24 Aug 18 17:08:38 localhost kernel: [20532.406080] sensitive data is diffused
to "tty1" (inode number: 1043, vcpu_id: 3) by "scp" (pid: 926)
```

図 5　scp コマンドを実行した際のカーネルログ

(1)　システムコール実行時間

既存の VMM における拡散追跡機能は，監視対象 VM が発行したすべてのシステムコールをフックし，機密情報の拡散追跡に関係するシステムコールに対応した追跡処理を行う．本研究では，これらの機密情報の追跡処理のうち，拡散情報管理表を参照・更新する処理を排他制御した．このため，追跡対象のシステムコールの処理時間にオーバヘッドが生じる可能性がある．

そこで，VMM における拡散追跡機能によるシステムコールの処理時間のオーバヘッドを明らかにするため，**表 3** の（ア）から（ウ）に示すそれぞれの計算機環境において，追跡対象のシステムコールの処理時間を測定した．例として，（イ）の計算機環境を**図 6** に示す．測定では，ゲスト 1 で測定対象のシステムコールを 102 回実行し，1 回目と 102 回目を除く 100 回の実行時間を記録し，100 回の測定結果中の最大値，最小値，平均値，中央値，および不偏分散を算出した．また，割り当てた vCPU が 2 個の場合は，一方の vCPU 上で，測定対象のシステムコールを約 1 分間発行するプログラムを実行した．

測定対象とするシステムコールは，ファイル操作に関するシステムコールである read システムコールと write システムコールである．また，VMM における拡散追跡機能が追跡対象としていないシステムコールである getpid システムコールを測定対象とする．

read システムコールの測定は，管理対象ファイルと非管理対象ファイルのそれぞれについて，ファイルをオープンした後に，ファイルの先頭から 1 バイトを読み込む処理を 102 回行うプログラムを用いて実施した．

また，write システムコールの測定は，管理対象ファイルと非管理対象ファイルのそれぞれについて，ファイルをオープンした後に，ファイルの先頭から 256 バイト読み込み，別のファイルに 256 バイト書き出す処理を 102 回行うプログラムにより実施した．

さらに，getpid システムコールの測定は，syscall() 関数

表 3　vCPU と物理コアの対応関係（監視対象 VM が 1 台の場合）

		VM に割り当てた vCPU の数	
		1 個	2 個
物理コアの vCPU による共有	なし	(ア)	(イ)
	あり	N/A	(ウ)

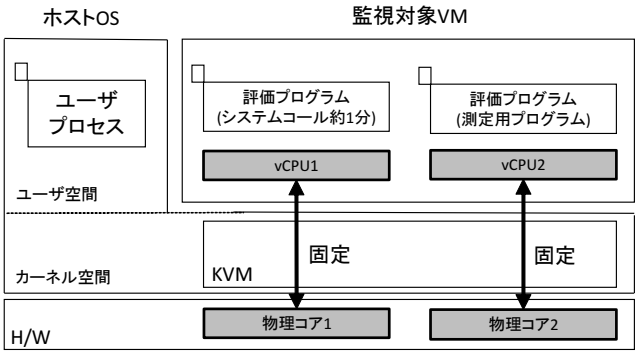

図 6　（イ）の計算機環境

表 4　システムコールの処理時間（μs）

		機能導入前（監視対象なし）	機能導入後（監視対象あり）								
			（ア）vCPU1 個，物理コア共有なし			（イ）vCPU2 個，物理コア共有なし			（ウ）vCPU2 個，物理コア共有あり		
			非管理対象資源の操作	管理対象資源の操作	オーバヘッド	非管理対象資源の操作	管理対象資源の操作	オーバヘッド	非管理対象資源の操作	管理対象資源の操作	オーバヘッド
getpid	最大値	0.34	26.27	-	25.93	41.66	-	41.32	19.53	-	19.19
	最小値	0.10	7.21	-	7.11	7.67	-	7.57	7.23	-	7.13
	平均値	0.10	7.67	-	7.57	9.44	-	9.34	7.28	-	7.18
	中央値	0.10	7.26	-	7.16	7.89	-	7.79	7.41	-	7.31
	分散	$5.85 * 10^{-3}$	6.15	-	-	27.10	-	-	1.50	-	-
read	最大値	0.37	28.37	29.98	29.61	26.59	25.26	24.89	25.92	49.03	48.66
	最小値	0.22	10.57	10.57	10.35	11.21	11.31	11.09	10.58	10.60	10.38
	平均値	0.24	10.87	11.07	10.83	11.66	11.66	11.42	10.82	11.12	10.88
	中央値	0.23	10.64	10.63	10.40	11.37	11.41	11.18	10.61	10.64	10.41
	分散	$4.05 * 10^{-4}$	3.33	6.05	-	2.76	2.23	-	2.48	15.28	-
write	最大値	1.10	150.82	133.83	132.73	145.57	244.82	243.72	130.72	137.90	136.80
	最小値	0.86	11.47	11.46	10.60	12.58	12.66	11.80	11.66	11.61	10.75
	平均値	0.87	18.72	18.97	18.10	21.01	22.79	21.92	17.92	19.70	18.83
	中央値	0.87	11.55	11.52	10.65	12.80	12.94	12.07	11.72	11.66	10.79
	分散	$9.35 * 10^{-4}$	738.46	788.30	-	942.24	1469.97	-	583.56	847.05	-

を介して getpid システムコールを発行する処理を 102 回行うプログラムにより実施した．これは，glibc の getpid() のラッパー関数は，プロセスが複数回 getpid() を呼び出した場合にその都度システムコールを呼び出すのを避けるため，PID をキャッシュするためである．

（2） bzImage のビルド時間

本評価では，システムコールが多数発行される bzImage のビルドに要する時間を測定した．これにより，管理対象ファイル管理表と管理対象プロセス管理表を排他制御した場合の VMM における拡散追跡機能による性能への影響を評価した．測定は，1 個の vCPU を割り当てた VM を用いた場合と 2 個の vCPU を割り当てた VM を用いた場合について行った．VMM における拡散追跡機能を導入した後の測定は，管理対象としたファイルがない場合とあらかじめ.config を管理対象ファイルとした場合について行った．.config を管理対象ファイルに設定して bzImage をビルドした際，.config が最初に読み込まれる．このため，その後に読み込まれるファイルやプロセスがすべて管理対象ファイルや管理対象プロセスに追加される．これにより，多くのファイルが管理対象となる場合の性能への影響を評価した．また，vCPU を 2 個割り当てた VM を用いた測定（(イ)，(ウ)）では，-j オプションを指定し，bzImage をビルドした処理時間を測定した．

4.3.2　システムコール実行時間の評価結果

表 3 の各計算機環境において，システムコールの実行時間を測定した結果を**表 4** に示す．表 4 において，「機能導入前（監視対象なし）」は，VMM における拡散追跡機能を導入していない場合，「機能導入後（監視対象あり）」は，VMM における拡散追跡機能を導入した後の場合を示す．また，「機能導入後（監視対象あり）」における「非管理対象資源の操作」と「管理対象資源の操作」の各項目は，それぞれ，非管理対象ファイルを操作する場合と管理対象ファイルを操作する場合であることを示す．「オーバヘッド」の項目は，getpid システムコールについては「非管理資源を操作した場合における処理時間 − 機能導入前における処理時間」により算出した値であり，read システムコールと write システムコールについては「管理対象資源を操作した場合における処理時間 - 機能導入前における処理時間」により算出した値である．

表 4 より，機密情報の拡散追跡機能の導入前と導入後の平均値を比較すると，read システムコールは最大 11.66μs，write システムコールは最大 21.92μs のオーバヘッドが生じていることが分かる．これは，これらのシステムコール処理における機密情報拡散の追跡処理によるものである．

また，機密情報の拡散に関係ないシステムコールである getpid システムコールの平均値を比較すると，オーバヘッドは最大で 9.34μs と比較的小さいことが分かる．機密情報の拡散に関係ないシステムコールをフックした際，機密情報の拡散追跡処理を行わず，制御をゲスト OS に戻す．このため，getpid システムコールのオーバヘッドは，システムコールの発行を検知し，検知したシステムコールが機密情報の拡散に関係するものが否か判定する処理によるものだと考えられる．

4.3.3　bzImage のビルド時間の評価結果

表 3 の各計算機環境において，bzImage のビルド時間を測定した結果を**表 5** に示す．表中の「機能導入前」の項目

表 5　bzImage のビルド時間の測定結果

			実行時間（s）
（ア）	機能導入前		159.55
	機能導入後	管理対象ファイルなし	188.31
		管理対象ファイルあり（.config）	188.97
		オーバヘッド（管理対象ファイルあり）	29.42
		オーバヘッドの割合（%）	18.44
（イ）	機能導入前		86.44
	機能導入後	管理対象ファイルなし	102.88
		管理対象ファイルあり（.config）	103.01
		オーバヘッド（管理対象ファイルあり）	16.57
		オーバヘッドの割合（%）	19.17
（ウ）	機能導入前		169.36
	機能導入後	管理対象ファイルなし	198.18
		管理対象ファイルあり（.config）	198.20
		オーバヘッド（管理対象ファイルあり）	28.84
		オーバヘッドの割合（%）	17.03

は，VMM における拡散追跡機能を導入していない場合の測定結果である．「機能導入後」における「管理対象ファイルなし」と「管理対象ファイルあり（.config）」の各項目は，VMM における拡散追跡機能を導入した場合の測定結果であり，それぞれ管理対象ファイルを登録していない場合と，あらかじめ.config を管理対象ファイルとして登録した場合の測定結果を示す．「オーバヘッド」の項目は，「あらかじめ.config を管理対象ファイルとして登録した場合の測定結果 − 機能導入前における測定結果」により算出した値である．「オーバヘッドの割合」は，「（あらかじめ.config を管理対象ファイルとして登録した場合の測定結果 / 機能導入前における測定結果）* 100」により算出した値である．

表 5 より，各計算機環境の実行時間において，17〜19%のオーバヘッドが生じていることが分かる．これは，bzImage のビルド中にシステムコールが多数発行されることにより，処理全体のオーバヘッドが大きくなるためだと考えられる．一方で，vCPU が 2 個で，かつ vCPU による物理コアの共有がない場合 (イ) において，管理対象ファイルを登録していない場合とあらかじめ.config を管理対象ファイルとして登録した場合の実行時間の差は，0.13s（0.13%）であった．このことから，各管理表を参照・更新する処理を排他制御した影響は小さいと考えられる．

5. 関連研究

VM 外部から VM 内部を監視する技術は VMI（Virtual Machine Introspection）[6] と呼ばれている．VMI を実現することを目的として，VMM と VM の間に存在するセマンティックギャップへ対処する研究が行われている．文献 [7] では，ゲスト OS で発行されたシステムコールをフックすることで，攻撃者が使用した認証情報や入力されたコマンドなどの情報を抽出する SSH ハニーポットが提案されている．VMM における拡散追跡機能では，監視対象 VM の情報をホスト OS 側で管理しているのに対し，文献 [7] で提案されているシステムは，VM 内の情報を別の VM で管理している．

6. おわりに

VMM における拡散追跡機能における複数の vCPU を割り当てた VM を監視対象とした場合の対応方法について述べた．既存の VMM における拡散追跡機能は，1 台の VM につき 1 つの vCPU にハードウェアブレークポイントを設置できる．これに対し，本研究では，1 台の VM に割り当てたすべての vCPU にハードウェアブレークポイントを設置するために，処理を変更した．また，監視対象 VM 管理表，管理対象ファイル，プロセス，ソケット，ポート番号管理表の整合性を保証するために，監視対象 VM 管理表，管理対象ファイル，プロセス，ソケット，ポート番号管理表を参照・更新する処理を排他制御した．さらに，VMM，ゲスト OS としてそれぞれ KVM，Linux を用いた環境において，VMM における拡散追跡機能の性能への影響を評価した結果を述べた．

謝辞　本研究の一部は，JSPS 科研費 JP19H04109 の助成を受けたものです．

参考文献

[1] 特定非営利活動法人 日本ネットワークセキュリティ協会（JNSA）：2018 年 情報セキュリティインシデントに関する調査報告書【速報版】，入手先 〈https://www.jnsa.org/result/incident/2018.html/〉（参照 2021-1-25）．

[2] 田端利宏，箱守　聰，大橋　慶，植村晋一郎，横山和俊，谷口秀夫：機密情報の拡散追跡機能による情報漏えいの防止機構，情報処理学会論文誌，Vol.50，No.9，pp.2088–2102（2009）．

[3] Fujii, S., Sato, M., Yamauchi, T., and Taniguchi, H.: Evaluation and Design of Function for Tracing Diffusion of Classified Information for File Operations with KVM, *The Journal of Supercomputing*, Vol.72, Issue 5, pp.1841–1861 (2016).

[4] Fujii, S., Sato, M., Yamauchi, T., and Taniguchi, H.: Design of Function for Tracing Diffusion of Classified Information for IPC on KVM, *Journal of Information Processing*, Vol.24, No.5, pp.1841–1861 (2016).

[5] 岡崎俊樹，森山英明，山内利宏，佐藤将也，谷口秀夫：VM 上の複数 VM の動作に対応した機密情報の拡散追跡機能，コンピュータセキュリティシンポジウム 2017（CSS2017）論文集, Vol.2017, No.2, pp.1295–1301（2017）．

[6] Nance K., Bishop M., Hay B.: Virtual Machine Introspection: Observation or Interference?, *IEEE Security & Privacy Magazine*, Vol.6, No.5, pp.32–37 (2008).

[7] Sentanoe S., Taubmann B., and Reiser H.P.: Virtual Machine Introspection Based SSH Honeypot, *Proc. 4th Workshop on Security in Highly Connected IT Systems*, pp.13–18 (2017).