Computer Security Symposium 2017
23 - 25 October 2017

KVM上の複数VMの動作に対応した機密情報の拡散追跡機能

岡﨑 俊樹[1]　森山 英明[2]　山内 利宏[1,a)]　佐藤 将也[1]　谷口 秀夫[1]

概要：我々は，仮想マシンモニタであるKVMを用いて，ゲストOS上で機密情報が拡散する経路を追跡する機能を提案している．具体的には，機密情報の拡散経路のうち，1台のVMが動作する場合について，ファイル操作，プロセス生成，およびソケット通信による機密情報の拡散を追跡する機能を実現している．そこで，本稿では，監視対象のVMが複数である場合について，機密情報の拡散をKVM上で追跡する機能を設計し，実現方式について述べる．具体的には，システムコールを発行したVMを判別し，VMごとに拡散情報を管理する方式について述べる．また，提案方式の追跡可能性とオーバヘッドを評価した結果を報告する．

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

Toshiki Okazaki[1]　Hideaki Moriyama[2]　Toshihiro Yamauchi[1,a)]　Masaya Sato[1]
Hideo Taniguchi[1]

1　はじめに

計算機上で機密情報を扱う機会の増加とともに，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，企業や個人にとって大きな損失となり，情報漏えいの防止が重要な課題となっている．個人情報漏えいインシデントの分析結果 [1] によると，管理ミスと誤操作が個人情報漏えいの原因の約50%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内部の機密情報の利用状況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，機密情報が拡散す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する状況を追跡し，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．また，機密情報の拡散追跡機能を拡張し，機密情報の拡散経路を可視化する機能 [3] と複数計算機間の機密情報拡散を追跡する機能 [4] が実現されている．

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に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．

そこで，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（以降，VMMにおける拡散追跡機能）が提案されている [5], [6]．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，VMからVMMの機能を能動的に呼び出すことは不可能である．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，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を困難とし，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，ホストOSとゲストOSのソー

[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] 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
[a)] yamauchi@cs.okayama-u.ac.jp

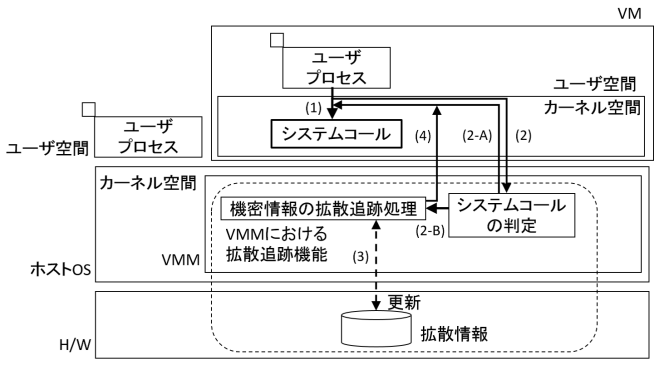

図 1 VMM における拡散追跡機能の全体像

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．

しかし，VMM における拡散追跡機能には，複数 VM の動作に対応していないという問題点がある．このため，複数 VM の動作への対応を検討する必要がある．

そこで，本稿では，複数台の監視対象の VM が動作した際に，機密情報の拡散を KVM 上で追跡する機能の設計と実現方式を述べる．また，複数の VM が動作した際の機密情報拡散の VMM における追跡可能性と性能評価を述べる．

2 VMM における機密情報の拡散追跡機能

文献 [5]，[6] で実現されている VMM における拡散追跡機能とその問題点について述べる．

2.1 機密情報の拡散経路

文献 [5]，[6] 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と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する．

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機能を 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像を**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 問題点

文献 [5] では，監視対象の VM が 1 台のみである場合について，(1) ファイル操作と (2) 子プロセス生成による機密情報の拡散を追跡する手法を述べた．また，文献 [6] では，監視対象の VM が 1 台のみである場合について，(3) プロセス間通信の手法の 1 つであるソケット通信による機密情報の拡散を追跡する手法を述べた．一方で，クラウドサービスの実現など，実際の VM の利用を考えた場合，1 つの計算機上で複数の VM が動作する状況は多く，複数の VM 内に機密情報が散在する状況が想定される．このため，監視対象の VM が複数台の場合において，VM の判別を行い，各 VM の機密情報の拡散・漏えいを追跡する機構を提案する．

3 複数 VM に対応した機密情報の拡散追跡機能の設計

3.1 追跡対象

2.1 節で述べたように，機密情報は，ファイル操作，子プロセス生成，およびプロセス間通信により拡散する．このため，上記の操作に関連するシステムコールをフックし，追跡処理を行う．以降の節では，複数の監視対象の VM が存在する場合において，各 VM 内におけるファイル操作，子プロセス生成，およびソケット通信による機密情報の拡散を VMM から追跡する手法の設計を述べる．

3.2 課題

監視対象の VM が複数である場合において，VMM における拡散追跡機能を用いて，機密情報の拡散を追跡し，漏えいを検知するには，以下の課題を解決する必要がある．

(課題 1)VM の判別

監視対象のすべての VM において発行されるシステム

監視対象VM管理表

0	VM_0のUUID	VM_0のkvm構造体のアドレス
1	VM_1のUUID	VM_1のkvm構造体のアドレス
2	VM_2のUUID	VM_2のkvm構造体のアドレス
	⋮	⋮
n-1	VM_{n-1}のUUID	VM_{n-1}のkvm構造体のアドレス

図 2 監視対象 VM 管理表

コールの発行を VMM 上で検知し，処理をフックする際，システムコールを発行した VM を VMM 上で判別する必要がある．ここで，VMM における機密情報の拡散追跡機能は，VMM から取得可能な情報のみを用いて VM を判別する必要がある．

(課題 2)VM ごとの拡散情報の管理

VMM における拡散追跡機能は，拡散情報として，機密情報を有する可能性のあるファイル，プロセス，およびソケットの情報を管理している．このため，複数 VM への対応にともない，VMM において，VM ごとに拡散情報を管理する必要がある．

3.3 対処

3.3.1 VM の判別

VM の判別に識別子を用いる．識別子は，VM ごとに必ずユニークな値である必要がある．VMM において参照可能な値のうち，VM ごとに一意となる値を以下に示す．

(1) VM の UUID

(2) kvm 構造体のアドレス

(1)VM の UUID は，BIOS 内に格納されている値であり，VM を再起動した場合や VM のコピーを作成した場合においても一意の値となる．BIOS 情報は，VMM において，SMBIOS 情報記述子を用いて参照することが可能であるため，物理アドレスを用いて直接メモリを参照することで，VMM における値の参照が可能である．

(2)kvm 構造体のアドレスは，VMM 上での参照が容易な値である．kvm 構造体は，VM ごとの vCPU ごとに定義され，VM の情報を保持する役割を持つ構造体である．ただし，VM の起動ごとに値は変化する．

上記の内容により，UUID は変化しない値であるため，kvm 構造体のアドレスよりも識別子として適切であるといえる．一方で，UUID の参照には BIOS 情報を用いるため，システムコールの発行ごとに UUID を取得して VM の判別を行う場合，UUID の取得時間による性能低下の問題がある．このため，VMM における UUID の参照回数を最小限に抑える必要がある．

そこで，UUID と kvm 構造体のアドレスを対応付けて識別子として用いるテーブルである監視対象 VM 管理表を実現する．監視対象 VM 管理表の実現イメージ図を**図 2** に示す．監視対象 VM 管理表は，VMM における拡散追跡機能の起動と同時に構成する．このため，VMM における拡散追跡機能の起動後は，kvm 構造体のアドレスを用いて，システムコールを発行した VM の判別が可能となる．また，UUID の参照回数は，監視対象の VM に対して 1 回のみでよいため，VMM における UUID の参照回数を最小限に抑えることができる．

VM_0 の 管理対象ファイル管理表

0	$inode_0$	$inode_0$のファイルパス名
1	$inode_1$	$inode_1$のファイルパス名
2	$inode_2$	$inode_2$のファイルパス名
	⋮	⋮

⋮

VM_{n-1} の 管理対象ファイル管理表

0	$inode_a$	$inode_a$のファイルパス名
1	$inode_b$	$inode_b$のファイルパス名
2	$inode_c$	$inode_c$のファイルパス名
	⋮	⋮

図 3 管理対象ファイル管理表

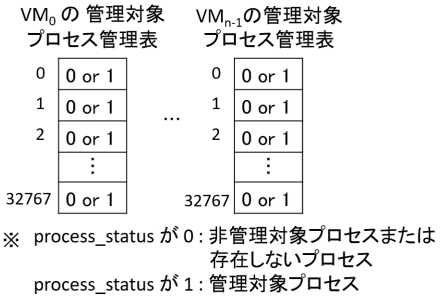

図 4 管理対象プロセス管理表

3.3.2 VM ごとの拡散情報の管理

VMM における拡散追跡機能は，拡散情報として，機密情報を有する可能性のあるファイル（以降，管理対象ファイル）と機密情報を有する可能性のあるプロセス（以降，管理対象プロセス）を管理している．具体的には，**図 3** に示すように，inode を用いて管理対象ファイル管理表を構成することで管理対象ファイルを管理し，**図 4** に示すように，PID を用いて管理対象プロセス管理表を構成することで管理対象プロセスを管理している．1 台の VM 内において，inode と PID は，必ず一意な値となる．一方で，VM をまたぐと，inode と PID は，必ずしも一意な値とはならない．このため，複数 VM への対応にともない，図 2 に示した監視対象 VM 管理表のインデックスごとに管理対象ファイル管理表と管理対象プロセス管理表を構成することで，監視対象の VM ごとに inode と PID を管理する機構を実現する．

また，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プロセスを操作したソケット（以降，管理対象ソケット）の情報を管理している．具体的には，UNIX ドメインの通信においては，ソケットのアドレスを用いて管理対象 UNIX ソケット管理表を構成することで管理対象

ソケットを管理し，INET ドメインの通信においては，ソケットに割り当てられたポート番号と送信先 IP アドレスを用いて管理対象 INET ソケット管理表を構成することで管理対象ソケットを管理している．このため，複数 VM への対応にともない，管理対象ソケットの情報について，監視対象の VM ごとにソケットの情報を管理する機構を実現する．

4　実現方式と評価

4.1　実現方式

4.1.1　実現における課題

3 章で述べた複数 VM に対応した機密情報の拡散追跡機能を kvm-kmod-3.9 に実現した．なお，今回の実現において，VMM における拡散追跡機能は，VM におけるシステムコールの発行には SYSCALL 命令と SYSRET 命令を利用することを前提とする．また，Intel 社の仮想化支援機能である VT-x の利用を前提とする．

3.3 節の設計をふまえて，実現における課題を以下に示す．

(実現課題 1)システムコールのフック

機密情報の拡散を漏れなく追跡するため，監視対象のすべての VM で発行されたシステムコールを VMM においてフックする方法を検討する必要がある．

(実現課題 2)VMM における UUID の取得

物理アドレスを用いて直接メモリを参照することで，VMM において UUID を取得する方法を検討する必要がある．

4.1.2　システムコールのフック

VMM における拡散追跡機能は，DR レジスタを用いて，SYSCALL 命令と SYSRET 命令にハードウェアブレークポイントを設置することで，デバッグ例外を発生させ，ゲスト OS から VMM へ処理を移行し，VM で発行されたシステムコールを VMM 上でフックしている．KVM 上で動作する VM の vCPU に関する情報は，kvm_vcpu 構造体に格納されている．ここで，kvm_vcpu 構造体から kvm_vcpu_arch 構造体を参照し，DR レジスタの値を書き換えることで，任意のアドレスに格納された命令 (SYSCALL 命令と SYSRET 命令) にハードウェアブレークポイントを設置することができる．また，kvm_vcpu 構造体は，各 vCPU に対して 1 つ定義される．このため，監視対象の VM のすべての vCPU において，vCPU に対応した DR レジスタの値を書き換えることで，監視対象の VM で発行されたシステムコールを VMM 上でフックすることができる．

4.1.3　VMM における UUID の取得

UUID は，BIOS 内に 16 バイトの値として格納されている．また，BIOS 内のデータ構造の配置は，SMBIOS(System Management BIOS) として定められている．BIOS 情報を格納しているブロック (BIOS Information with strings) の先頭アドレス (Structure Table Address) は，ゲスト物理アドレスの特定の範囲内に存在する Entry Point Structure 内に保持されている [9]．このため，**図 5** に示すように，VMM においてゲスト物理メモリを参照し，Entry Point Structure を探索することで，BIOS Information with strings の内容を参照できる．

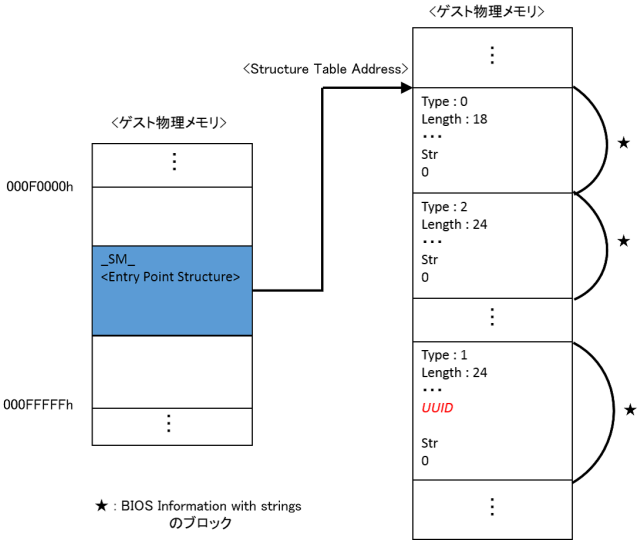

図 5　Entry Point Structure

表 1　ホスト OS の評価環境

OS	Fedora 18 (Linux 3.6.10, 64bit)
CPU	Intel(R) Xeon E5-2609V4(8 コア)
メモリ	64GB
VMM	kvm-kmod-3.9

表 2　ゲスト OS の評価環境

OS	Fedora 18 (Linux 3.6.10, 64bit)
仮想 CPU 数	1
メモリ	1024MB

BIOS Information with strings は，保持する BIOS 情報の属性によって 0 から 255 の Type として分類されており，UUID は Type1 のブロック内に保持されている．このため，Structure Table Address をもとに，Type が 1 であるブロックを探索することで，VMM から VM の UUID を参照できる．

4.2　評価内容と評価環境

提案方式の有用性と提案方式の適用におけるオーバヘッドを明確にするために，**表 1**，**表 2** に示す環境において，以下の評価を行った．

(評価 1)機密情報拡散に対する追跡可能性

監視対象の VM が複数である場合において，機密情報の拡散・漏えいを検知できるか否かについて検証し，提案方式の有用性を示す．

(評価 2)提案方式によるオーバヘッド量

監視対象の VM が複数である場合において，提案方式の適用における性能への影響を明らかにするため，UUID の取得時間とシステムコールの実行時間を測定した．

4.3 追跡可能性

4.3.1 評価方法

監視対象の VM が 2 台存在し，いずれの VM においても機密情報を有するファイルが 1 つずつ存在している場合において，以下に示す 2 つの事例を用いて機密情報の拡散と漏えいの追跡可能性を検証した．

(事例 1)ファイル操作と子プロセス生成による機密情報拡散

監視対象のそれぞれの VM において，管理対象ファイルを読み込んだ後に，その内容を書き出し先のファイルに出力した後に，管理対象ファイルを読み込んだプロセスの子プロセスを生成する．その後，監視対象のすべての VM の管理対象ファイル管理表と管理対象プロセス管理表を確認し，ファイル操作と子プロセス生成による機密情報の拡散を追跡できているか否かを評価する．

(事例 2)ローカルプロセス間通信による機密情報拡散

監視対象のそれぞれの VM において，クライアント側で管理対象ファイルを読み込み，サーバ側にそのファイル内容を送信する．その後，監視対象の VM のすべての管理対象プロセス管理表を確認し，ローカルプロセス間通信による機密情報の拡散を追跡できているか否かを検証する．

4.3.2 評価結果

ファイル操作と子プロセス生成においては，監視対象の各 VM 内において，管理対象ファイルを読み込んだプロセスが，書き出し先のファイルにその内容を出力した時点で機密情報が拡散する．さらに，管理対象ファイルを読み込んだプロセスが子プロセスを生成した際に，子プロセスへと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，プロセスが機密情報ファイルを読み込む read() をフックした後に read() を発行した VM を判別し，対応する VM の管理対象プロセス管理表に対し，機密情報を読み込んだプロセスの PID を追加した．また，機密情報を読み込んだプロセスが機密情報を書き出した先のファイルの inode を，対応する VM の管理対象ファイル管理表に追加した．さらに，機密情報を読み込んだプロセスが生成した子プロセスの PID を，対応する VM の管理対象プロセス管理表に追加した．

また，ローカルプロセス間通信においては，監視対象の各 VM 内において，送信側の管理対象プロセスがソケットに機密情報を送信した時点でソケットに機密情報が拡散する．その後，受信側のプロセスがソケットから機密情報を受信することにより，受信側のプロセスへ機密情報が拡散する．（事例 2）を行った際，VMM における拡散追跡機能は，ソケットに機密情報を送信する sendto() をフックした後に sendto() を発行した VM を判別し，ソケットへの機密情報の拡散を検知し，ソケットのアドレスを対応する VM の管理対象ソケット管理表に追加した．また，ソケットから機密情報を受信する recvfrom() をフックした後に recvfrom() を発行した VM を判別し，ソケットからプロセスへの機密情報の拡散を検知し，対応する VM の管理対象プロセス管理表に対し，機密情報を受信したプロセスの PID を追加した．

4.4 オーバヘッド

4.4.1 評価方法

評価項目は以下の通りである．測定は，表 1 および表 2 に示した環境において行った．

(1) UUID 取得時間

3.3 節で述べた通り，UUID を取得する処理において，BIOS 情報を参照し，UUID が格納されているブロックを探索する処理を行うため，性能低下が懸念される．このため，UUID の取得に要する時間を測定した．

(2) システムコール実行時間

提案方式では，すべてのシステムコールをフックし，機密情報の拡散に関係するシステムコール処理において追跡処理を行うため，性能低下が懸念される．そこで，監視対象の VM が 2 台存在する場合において，提案方式の適用によるオーバヘッドを明らかにするために，システムコールの実行時間を測定した．ファイル操作に関する read と write，子プロセス生成に関する fork，ソケット通信に関する sendto と recvfrom を測定対象とした．また，比較対象として，機密情報の拡散に関係しないシステムコールである getpid を測定対象とした．

4.4.2 評価結果

監視対象の VM が 2 台存在する場合において，VMM 上での UUID 取得時間は，271.06μs であった．UUID の取得回数は，監視対象の VM の数に比例する．しかし，各 VM の起動時に 1 回行うだけなので，処理に与える影響は小さい．

次に，監視対象の VM が 2 台存在する場合におけるシステムコール実行時間の評価結果について，**表 3** に示す．表 3 の「機能導入前」の項目は，VMM における拡散追跡機能を導入していない環境での測定値である．また，「機能導入後」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，VMM における拡散追跡機能を導入した環境での測定値であり，それぞれ管理対象ファ

表 3 システムコールの処理時間（μs）

	機能導入前	機能導入後		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	非管理対象資源の操作	管理対象資源の操作
read(file)	9.26	21.96	22.24	12.70	12.97
write(file)	8.49	21.02	31.64	12.53	23.15
fork	149.79	180.60	193.86	30.80	44.07
sendto	33.00	76.04	88.22	43.03	55.22
recvfrom	32.73	62.28	66.53	29.54	33.80
getpid	0.16	7.86	-	7.70	-

イルを操作しない場合と管理対象ファイルを操作する場合における測定値である．「オーバヘッド」の項目は，「機能導入後における測定値 - 機能導入前における測定値」により算出した値である．

機密情報の拡散に関係するシステムコールである read，write，fork，sendto，および recvfrom にはそれぞれ，非管理対象資源の操作の場合で 12.70μs，12.53μs，30.80μs，43.03μs，29.54μs，管理対象資源操作の場合で 12.97μs，23.15μs，44.07μs，55.22μs，33.80μs のオーバヘッドが生じていることがわかる．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，管理対象資源の操作の場合の方が，0.27μs から 13.27μs だけオーバヘッドが大きい．関係するシステムコール当たり数十 μs のオーバヘッドのため，アプリケーション処理に与える影響は，小さいと推察できる．

また，機密情報の拡散に関係しないシステムコールである getpid のオーバヘッドは，7.70μs と比較的小さい値である．getpid は，カーネル内での処理時間が短く，測定結果における実際の処理時間の割合が小さいシステムコールである．このため，getpid のオーバヘッドは，システムコールに対するフック機構自体のオーバヘッドとみなすことができる．

以上より，提案方式において，システムコールをフックし，システムコールを発行した VM を判別する機構自体の性能への影響は，小さいといえる．

5 関連研究

機密情報漏えいの防止を目的とした研究に，Filesafe[7] がある．Filesafe は，あらかじめ，機密情報を有するファイルに対して，read/write システムコールの実行可否を設定する．その後，ゲスト OS で発行された read/write システムコールを VMM でフックし，ユーザの設定に違反している操作の場合は操作を中断させることにより，ユーザの意図しない機密情報の漏えいを防止する．一方で，VMM における拡散追跡機能は，Filesafe のように機密情報を有するファイルを逐一設定する必要はなく，機密情報の拡散を自動で追跡し，機密情報を有するファイルを管理するため，ユーザの設定漏れによる漏えいが発生する可能性は低い．

また，Aquifer[8] は，モバイル端末において，ポリシを用いて，機密情報を扱うことのできるアプリケーションを制限することにより，ユーザの意図しない情報漏えいを防止する．Aquifer は，スマートフォンの OS 内部に実現されており，端末内の機密情報の拡散を追跡する．一方で，VMM における拡散追跡機能は，ホスト OS とゲスト OS のソースコードを改変することなく，VMM のみの改変により実現しているため，より多くの環境への導入が可能である．

6 おわりに

監視対象の VM が複数存在する場合において，VMM において機密情報の拡散を追跡する機能の実現方式を述べた．具体的には，UUID と kvm 構造体を用いてシステムコールを発行した VM を VMM において判別し，拡散情報として，機密情報を有する可能性のあるファイル情報，プロセス情報，およびソケット情報を VM ごとに管理することで，機密情報の拡散を追跡する機能の実現方式について述べた．VM の判別には，VM ごとにユニークな値であり，VMM 上から取得可能な値である，VM の UUID と kvm 構造体のアドレスを用いる．また，拡散情報を VM ごとに管理することで，監視対象のすべての VM に対して漏れなく機密情報の拡散を追跡することができる．

また，VMM に KVM，ゲスト OS に Linux を用いて実装し，評価した結果を述べた．事例を用いた評価では，監視対象の VM が複数の場合において，ファイル操作，子プロセス生成，ソケット通信による機密情報拡散の追跡が VMM 上で可能であることを示した．性能評価では，監視対象の VM が複数の場合において，UUID 取得時間とシステムコール実行時間を測定し，システムコールのフック処理と VM の判別処理の性能への影響が小さいことを示した．

謝辞 本研究の一部は，科学研究費補助金基盤研究 (B)（課題番号：16H02829）による．

参考文献

[1] 日本ネットワークセキュリティ協会：2016年情報セキュリティインシデント調査結果, http://www.jnsa.org/result/incident/.

[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] 福島健太，山内利宏，谷口秀夫：可視化とフィルタリング機能により機密情報の拡散追跡を支援する機構の実現，情報処理学会論文誌，Vol.53，No.9，pp.2171–2181（2012）.

[4] Otsubo, N., Uemura, S., Yamauchi, T., and Taniguchi, H.: Design and Evaluation of a Diffusion Tracing Function for Classified Information Among Multiple Computers, *7th FTRA International Conference on Multimedia and Ubiquitous Engineering (MUE 2013)*, pp.235–242 (2013).

[5] 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, no.5, pp.1841–1861 Springer (2016).

[6] 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).

[7] Wang, J., Yu, M., Li Bingyu, Q.Z., and Guan, H.: Hypervisor-based Protection of Sensitive Files in a Compromised System, *Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC 2012)*, pp.1765–1770 (2012).

[8] Nadkarni, A. and Enck, W.: Preventing accidental data disclosure in modern operating systems, *Proceedings of the 2013 ACM SIGSAC conference on Computer and communications security (CCS 2013)*, pp.1029–1042 (2013).

[9] System Management BIOS (SMBIOS) Reference Specification Version 3.1.1, https://www.dmtf.org/sites/default/files/standards/documents/DSP0134_3.1.1.pdf.