Vol.2014-CSEC-66 No.28
Vol.2014-SPT-10 No.28
2014/7/3

KVMにおける機密情報の拡散追跡機能の設計

藤井 翔太[1,a)]　山内 利宏[1]　谷口 秀夫[1]

概要：計算機内の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として機密情報の拡散追跡機能を OS 内に実現した．しかし，機密情報の拡散追跡機能には，導入の際に対象の OS のソースコードを修正する必要があること，および OS 自体を攻撃されると，機密情報の拡散追跡機能が無効化される危険性が存在する問題がある．そこで，KVM における機密情報の拡散追跡機能を設計した．機密情報の拡散追跡機能を KVM 内に実現することにより，導入対象 OS のソースコードを修正する必要がなくなり，より多くの環境に導入可能になる．また，機密情報の拡散追跡機能を OS から隔離できるため，機密情報の拡散追跡機能への攻撃をより困難にできる．

1. はじめに

計算機の性能向上と普及により，様々なサービスにおいて，顧客情報などの機密性の高い情報を計算機で扱う機会が増加している．機密性の高い情報は，その情報の一部が外部へ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になる．個人情報漏えいのインシデントの分析結果 [1] によると，管理ミスと誤操作は，情報漏えい原因全体の約 73%を占めている．このような情報漏えいを防止するには，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，機密情報が計算機内に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と呼ぶ），および機密情報が漏えいする可能性を有する処理を制御することにより，未然に情報漏えいを防止する機能（以降，書き出し制御機能と呼ぶ）を実現した [2]．しかし，機密情報の拡散追跡機能を導入する場合，対象のオペレーティングシステム（以降，導入対象 OS と呼ぶ）のソースコードを修正する必要があるため，Windows などのソースコードが公開されていない OS には導入できない．また，カーネルのバージョンを変更する場合，機密情報の拡散追跡機能のソースコードを別のバージョン向けに修正する必要がある．

また，セキュリティソフトが攻撃者に乗っ取られる可能性が指摘されている [3]．既存の機密情報の拡散追跡機能（以降，既存機能と呼ぶ）は，OS 内部に実現されており，機密情報を搾取しようとする攻撃者や悪意のある計算機の利用者に攻撃を受けて機能を無効化された場合，情報漏えいの防止や漏えい経路の把握が困難になる．

そこで，本稿では，仮想計算機モニタ（Virtual Machine Monitor，以降，VMM と呼ぶ）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と呼ぶ）の設計を述べる．また，VMM における拡散追跡機能は，ゲスト OS のソースコードを改変することなく，VMM の改変により，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する．VMM は，OS よりも攻撃が困難である [4] ため，VMM における拡散追跡機能への攻撃がより困難になると期待できる．本稿では，VMM として KVM (Kernel-based Virtual Machine [5]) を用いた場合の設計を述べる．

2. 機密情報の拡散追跡機能

2.1 基本機構

文献 [2] における機密情報の拡散追跡機能の基本機構を**図 1** に示し，以下で処理の流れを説明する．

(1) 機密情報の拡散に関係するシステムコールをフック
(2) 情報収集処理では，機密情報の拡散追跡に必要な情報（読み込むファイルや通信先のプロセスなどの情報）を収集
(3) 更新・検査処理では，取得した情報をもとに拡散情報（管理対象ファイルや管理対象プロセスの情報）を更新し，機密情報が漏えいする可能性を検査
　(A) 検査により機密情報が漏えいする可能性が発見された場合，監視応用プログラム（監視 AP）へ通知
　(B) 検査により機密情報が漏えいする可能性が発見されなかった場合，システムコールのフック元へ復帰
(4) 監視 AP から利用者の判断結果を受け取った後，利用

[1] 岡山大学 大学院自然科学研究科
[a)] fujii@swlab.cs.okayama-u.ac.jp

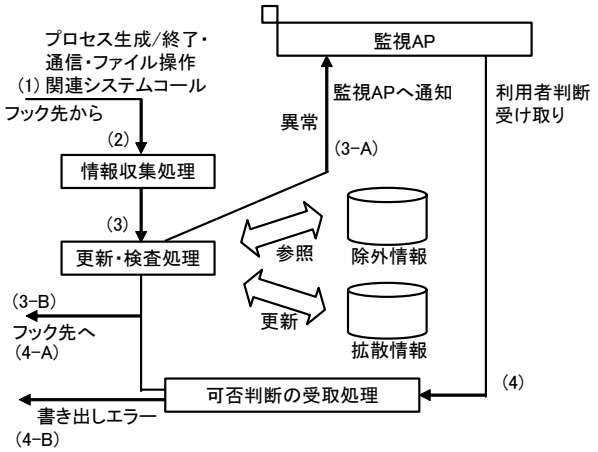

図 1 機密情報の拡散追跡機能の基本機構

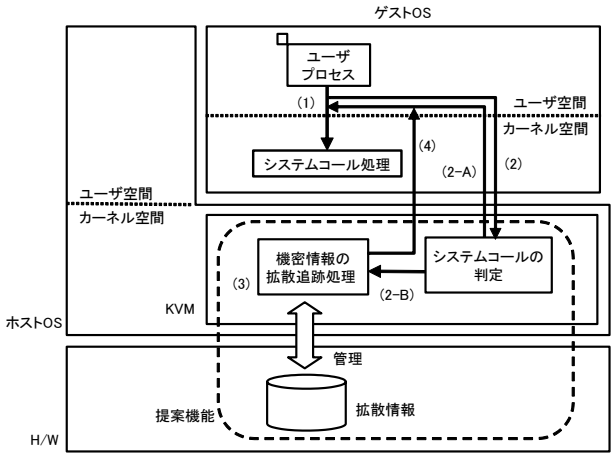

図 2 VMM における拡散追跡機能の全体図

者の判断に従い，システムコール処理を制御
(A) 利用者判断が実行可能の場合，システムコール処理を継続
(B) 利用者判断が実行不可の場合，システムコール処理をエラーとして終了

また，既存機能は，機密情報の拡散が行われないと利用者が判断したファイルとプログラムを監視対象から除外する機能を有する．監視対象から除外するファイルとプログラムは，除外情報に登録する．

2.2 問題点

既存機能の問題点を以下に述べる．

(問題 1) 導入する際に OS のソースコードの修正が必要
機密情報の拡散追跡機能を導入する際，導入対象 OS のソースコードを修正する必要がある．このため，Windows などのソースコードが公開されていない OS への導入は難しい．また，既存機能は，特定のカーネルのバージョンにしか対応していない．このため，カーネルのバージョンを変更する場合，カーネルのバージョンに応じて，機密情報の拡散追跡機能のソースコードを修正する必要がある．

(問題 2) 機密情報の拡散追跡機能を無効化される危険性
既存機能は，OS 内部に実現している．このため，機密情報を搾取しようとする攻撃者や悪意のある計算機の利用者は，OS 自体を攻撃し，機密情報の拡散追跡機能を無効化できる．機密情報の拡散追跡機能が無効化された場合，情報漏えいの防止や漏えい経路の把握は困難になる．

本稿では，（問題 1）と（問題 2）に対処する方式を提案する．

3. KVM における機密情報の拡散追跡機能

3.1 VMM における拡散追跡機能への要件

2.2 節で述べた問題を解決するための要件を以下に示す．

(要件 1) 導入対象 OS のソースコードを修正する必要のない機密情報の拡散追跡機能の実現
（問題 1）への対処として，OS のソースコードを修正することなく導入できる機密情報の拡散追跡機能の実現が挙げられる．これにより，Windows などのソースコードが公開されていない環境にも導入可能になる．

(要件 2) 機密情報の拡散追跡機能の OS からの隔離
（問題 2）への対処として，機密情報の拡散追跡機能を導入対象 OS から隔離することが挙げられる．これにより，機密情報を搾取しようとする攻撃者や悪意のある計算機の利用者による機密情報の拡散追跡機能への攻撃をより困難にする．

3.2 基本機構

VMM における拡散追跡機能は，既存機能と同等の機能を VMM に実現するものである．VMM における拡散追跡機能の全体図を**図 2** 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，ゲスト OS のソースコードを改変することなく，ゲスト OS に対して既存機能と同等の機能を提供する．

3.3 仮想化の方式

VMM における拡散追跡機能では，OS のソースコードの修正が不要であること，および VM から VMM の機能

を能動的に呼び出すことはできないことの利点から，準仮想化ではなく完全仮想化を用いる．VMM における拡散追跡機能は，完全仮想化により仮想化を実現する KVM を用いることにより，3.1 節で述べた各要件を満たす．ただし，VMM における拡散追跡機能は KVM 特有の機能を使用しない．このため，VMM として KVM 以外を使用する場合でも，VMM における拡散追跡機能の実現は可能である．

KVM は CPU の仮想化支援機能を利用して完全仮想化を実現する．このため，CPU は Intel 社の VT-x や AMD 社の AMD-V といった仮想化支援機能を必要とする．これらの仮想化支援機能は近年の多くの CPU に搭載されている．以降では，Intel 社の仮想化支援機能である VT-x の利用を前提とする．

3.4 課題

既存機能と同等の機能を KVM により実現するための課題には，以下の 2 つがある．

(課題 1) 仮想計算機モニタによるシステムコールの情報の取得

機密情報は，システムコールの発行を契機として拡散する．このため，発行されたシステムコールが機密情報に関係するかどうかを判定する情報を取得する必要がある．

(課題 2) 仮想計算機モニタによる OS の情報の取得

機密情報の拡散追跡機能は，機密情報を有する可能性のあるファイルやプロセスを管理する．このため，実行中のプロセスと通信先のプロセスの情報や実行中のプロセスが扱うファイルの情報などの OS の情報を取得する必要がある．このとき，VM と VMM 間のセマンティックギャップ [4] を解消する必要がある．

3.5 仮想計算機モニタによるシステムコールの情報の取得

3.5.1 システムコールのフック

KVM によりゲスト OS におけるシステムコールの発行を検知するために，システムコールの入口（sysenter）をフックする．sysenter はシステムコールを発行する際に実行される命令である．このため，sysenter をフックすることにより，システムコールの発行を検知できる．

また，システムコールの戻り値として，システムコール処理の成否やシステムコールを発行したプロセスが扱うファイルの情報などが返却される．このため，システムコールを発行したプロセスが扱うファイルを特定するために，システムコールの戻り値を取得する必要がある．また，システムコールの成否を取得することにより，偽陰性と偽陽性を抑制できる．

そこで，VMM における拡散追跡機能ではシステムコールの入口に加えて，システムコールの出口（sysexit）もフックする．sysexit はシステムコール処理の終了時に，カーネルモードからユーザモードへ復帰する命令である．システムコールの出口をフックすることにより，システムコールの戻り値が取得できる．

3.5.2 システムコールフック時に取得する情報

VMM における拡散追跡機能は，すべてのシステムコールの発行を検知し，処理をフックする．このとき，発行されたシステムコールが機密情報の拡散に関係するシステムコールか否か判定する必要がある．システムコールの判定にはシステムコール番号を利用する．

また，発行されたシステムコールが機密情報の拡散に関係するシステムコールだった場合，通信先のファイルやプロセスに機密情報が拡散する可能性がある．このため，通信先のファイルやプロセスを特定する必要がある．システムコールは引数にシステムコールが扱うファイルやプロセスの情報を取るため，システムコールの引数を取得することにより，通信先のプロセスやファイルを特定できる．たとえば，read() システムコールは，第一引数に，ファイルを識別するための識別子であるファイルディスクリプタ（以降，fd と呼ぶ）を取る．fd を利用することにより，ファイル名やファイルのサイズなどを取得できる．このため，システムコールの引数を取得する．

さらに，3.5.1 項で述べたように sysexit のフック時にシステムコールの戻り値を取得する．

なお，機密情報の拡散に関係に関係するシステムコールについては文献 [2] で述べられている．しかし，文献 [2] は Linux 2.6.0 で実現している．このため，Linux 2.6.0 以外の環境に VMM における拡散追跡機能を導入する場合，導入環境に合わせて機密情報の拡散に関係するシステムコールを調査する必要がある．

3.6 仮想計算機モニタによる OS の情報の取得

3.6.1 ファイルを識別するための情報の取得

機密情報の拡散追跡機能は，機密情報をファイル単位で管理する．このため，ファイルを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呼ぶ）として登録する際，プロセスが読み込むファイルが管理対象ファイルか否かを判定する際，および書き出し先のファイルを管理対象ファイルとして登録する際にファイルを一意に識別する情報を取得する必要がある．そこで，ファイルを一意に識別する情報として，ファイルの inode 番号を取得する．inode 番号は，ファイルごとに一意に与えられる識別子である．プロセスがファイルを読み込む際と書き込む際に，プロセスが扱うファイルの inode 番号を取得することにより，そのプロセスが管理対象ファイルを扱っているか否かを判定できる．

3.6.2 プロセスを識別するための情報の取得

プロセスが管理対象ファイルを読み込んだ際，そのプロセスは他のプロセスやファイルに機密情報を伝える可能性がある．このため，機密情報の拡散追跡機能は機密情報を

表 1　ゲスト OS 上の情報を取得する手法の利点と欠点

	利点	欠点
(手法 A)	情報の取得が容易	機構への攻撃が容易 OS での AP の動作が必要
(手法 B)	OS が機構を意識せず 機構への攻撃が困難	OS への依存度高 データ構造の解析が必要

表 2　実現環境

CPU	Intel(R) Core(TM) i5-3470
VMM	kvm-kmod-3.9
ゲスト OS	Debian 7.1.0-i386 (Linux 3.2.46, 32bit)
ホスト OS	Fedora 18 (Linux 3.6.10, 64bit)

有する可能性のあるプロセス（以降，管理対象プロセスと呼ぶ）を管理する．そこで，管理対象プロセスを管理するために，プロセスを一意に識別する情報を取得する必要がある．プロセスを一意に識別する情報として，プロセス ID（以降，PID と呼ぶ）を取得する．PID はプロセスごとに一意に与えられる識別子である．プロセスが管理対象ファイルを読み込む際にそのプロセスの PID を取得し，機密情報ファイルを読み込んだプロセスを管理対象プロセスに追加する．また，管理対象プロセスがプロセス間通信を行う際に通信先プロセスの PID を取得し，通信先プロセスを管理対象プロセスに追加する．

3.6.3　セマンティックギャップの解消

3.6.1 項と 3.6.2 項で述べた情報を KVM から取得する際，VM と VMM 間のセマンティックギャップを解決する必要がある．セマンティックギャップを解決する方法として，以下の二つが考えられる．

(手法 A)　ゲスト OS 上で応用プログラム (AP) を実行し，ゲスト OS の情報を取得後，KVM へ取得した情報を送信

(手法 B)　メモリ領域を参照し，OS の情報を直接取得

（手法 A）はゲスト OS 上で AP を実行するため，OS の情報を容易に取得可能である．しかし，機密情報の拡散追跡機能が KVM 内で完結しない．また，OS の情報を取得する機構をゲスト OS 上で実現する場合，KVM 内に実現する場合に比べ，攻撃者による改変を容易にする．

（手法 B）は KVM 側でゲスト OS が利用するメモリ領域を直接参照することにより OS の情報を取得する．このため，ゲスト OS は OS の情報を取得する機構を意識することはない．また，ゲスト OS の情報を取得する機構が KVM 内に実現できるため，攻撃者による改変も受けにくい．しかし，ゲスト OS のデータ構造を解析する必要がある．

（手法 A）と（手法 B）の利点と欠点をまとめ，**表 1** に示す．（手法 A）には，OS の情報を取得する機構への攻撃が容易であるという欠点がある．KVM 内に機密情報の拡散追跡機能を実現することの利点の 1 つに，機密情報の拡散追跡機能への攻撃がより困難になる点が挙げられる．このため，OS の情報を取得する機構や機密情報の拡散追跡機能に対して攻撃の手段を増やす（手法 A）の利用は好ましくない．一方，（手法 B）は，データ構造の解析が必要であるものの，KVM 内に OS の情報を取得する機構を実現できる．このため，（手法 A）に比べ，機構への攻撃がより困難である．以上より，ゲスト OS 上の情報を取得する手法には（手法 B）を採用する．

3.7　期待される効果

VMM における拡散追跡機能の導入により，以下の効果が期待できる．

(効果 1)　導入対象 OS のソースコードの修正が不要

OS 内に実現している既存機能とは異なり，カーネルのソースコードを修正できない Windows などの OS への VMM における拡散追跡機能の導入が期待できる．

(効果 2)　機密情報の拡散追跡機能への攻撃が困難

VMM における拡散追跡機能は，機密情報の拡散追跡機能を導入対象 OS から KVM 内に隔離している．これにより，機密情報を搾取しようとする攻撃者や悪意のある計算機の利用者は VMM における拡散追跡機能を攻撃することがより困難になる．

(効果 3)　カーネルのバージョンの変更に対応可能

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，KVM を改変することにより，実現する．このとき，VMM における拡散追跡機能の依存する情報には，以下の 2 つがある．

(1) システムコールの仕様

VMM における拡散追跡機能は，システムコール番号をもとに，どのシステムコールが発行されたか判定する．また，システムコールの引数を取得する．このため，VMM における拡散追跡機能は，システムコール番号の割り当てや引数など，システムコールの仕様に依存する．

(2) プロセス情報を扱う構造体のデータ構造

VMM における拡散追跡機能は，inode 番号や PID はゲスト OS のデータ構造を解析し，ゲスト OS の利用するメモリ領域から直接取得する．このため，VMM における拡散追跡機能は，ゲスト OS のデータ構造に依存する．

このため，導入対象 OS のシステムコールの仕様やデータ構造が変化しない限り，カーネルのバージョンを変更した場合でも VMM における拡散追跡機能を引き続き利用できる．

4.　実現方式

4.1　実現環境

実現環境を**表 2** に示す．VMM に KVM，ゲスト OS に Linux を使用する．また，システムコールは，sysenter により，発行されることを前提とする．VMM における拡散追跡機能は sysenter の発行を検知してシステムコールをフックし，sysexit の発行を検知して戻り値を取得する．システムコールは，sysenter により，発行されることを前提

表 3　システムコールの入口をフックする手法の利点と欠点

	利点	欠点
(手法 1)	メモリの書き換えが不要	sysenter のみフック可
(手法 2)	フック箇所を自由に設定可 メモリの書き換えが不要	設定数に制限あり
(手法 3)	フック箇所を自由に設定可 設定数に制限なし	メモリの書き換えが必要

とする．

4.2　仮想計算機モニタによるシステムコールの情報の取得

4.2.1　システムコールの入口をフックする手法

システムコールの入口をフックする手法には，以下の 3 つがある．

(手法 1)　`sysenter_eip_msr` の書き換え
(手法 2)　ハードウェアブレークポイントの利用
(手法 3)　ソフトウェアブレークポイントの利用

(手法 1) は文献 [6] で述べられている手法であり，`sysenter_eip_msr` をカーネル空間の未使用な領域に書き換えることにより，システムコール発行時にページフォルト例外を発生させ，KVM 側へ制御を移行する．(手法 1) はレジスタの値を書き換えるため，後述するソフトウェアブレークポイントとは違い，メモリを書き換える必要がない．

(手法 2) はシステムコールのエントリポイントにハードウェアブレークポイントを設置することによりシステムコールの入口をフックする．ハードウェアブレークポイントはレジスタにより指定するため，(手法 1) と同様，メモリを書き換える必要がない．また，フックしたい処理の格納されているアドレスが分かれば，自由な場所をフックできる．しかし，ハードウェアブレークポイントを指定するレジスタは 4 つしかないため，多用することができない．

(手法 3) はシステムコールのエントリポイントに int3 命令を埋め込むことによりシステムコールの入口をフックする．int3 命令を埋め込む位置は自由に指定可能であるため，フックしたい処理の格納されているアドレスが分かれば，自由な場所をフックできる．また，ブレークポイントの個数に制限がない．しかし，ソフトウェアブレークポイントを利用する場合，メモリを書き換える必要がある．

上に示した (手法 1) ～ (手法 3) の利点と欠点をまとめ，**表 3** に示す．(手法 3) は (手法 1) や (手法 2) とは異なり，メモリの書き換えが必要であるという欠点がある．ゲスト OS を改変したくないため，メモリ領域の書き換えが必要な (手法 3) の利用は好ましくない．また，(手法 2) はメモリの書き換えは不要であるものの，設置可能数が 4 つと少ないため，積極的な使用はできない．一方，(手法 1) はメモリの書き換えが不要であり，sysenter のみフック可能という欠点もシステムコールの入口をフックする際には問題にならない．以上より，システムコールの入口のフックには (手法 1) を採用する．

4.2.2　システムコールの出口をフックする手法

システムコールの出口（sysexit）には sysenter における `sysenter_eip_msr` に相当するレジスタは存在しない．このため，(手法 1) を利用出来ない．そこで，sysexit が格納されているメモリアドレスにブレークポイントを設置することにより，処理をフックする．また，ブレークポイントには，メモリを書き換える必要のないハードウェアブレークポイントを用いる．

以上より，システムコールの出口のフックには 4.2.1 項の (手法 2) を採用する．このとき，デバッグ例外を発生させるためにデバッグ制御レジスタ（`DR7`）で sysexit が格納されたアドレスを指定したブレークポイント・アドレス・レジスタを有効化しておく必要がある．

また，デバッグ例外発生時に VM Exit を発生させるために Exception Bitmap [7] を利用する．例外発生時に，CPU により，Exception Bitmap の発生した割り込みの割り込み番号に対応するビットが参照される．このとき，対応するビットが 1 なら VM Exit を発生させて KVM で，ビットが 0 ならゲスト OS で処理される．このため，Exception Bitmap により，デバッグ例外発生時に処理を KVM へ移行できる．

4.2.3　システムコール番号，引数，および戻り値を取得する手法

システムコール番号は，システムコール発行時に `EAX` レジスタに格納される．`EAX` レジスタの値は VM Exit 時に Virtual Machine Control Structure（以降，VMCS と呼ぶ）に退避される．このため，KVM から VMCS を参照することにより取得できる．また，システムコールの引数は第 1 引数から順に `EBX` レジスタ以下の汎用レジスタに格納される．このため，システムコール番号と同様に，KVM から VMCS を参照することによって取得できる．

システムコールの戻り値はシステムコール処理の返却時に `EAX` レジスタに格納される．このため，システムコールの出口のフック時に KVM から VMCS を参照することにより取得できる．

4.3　仮想計算機モニタによる OS の情報の取得

システムコール処理をフックした際，実行中のプロセスの PID と実行中のプロセスが扱うファイルの inode 番号を取得する．inode 番号と PID はハードウェアの情報ではないため，システムコール番号や引数とは違い，VMCS に退避されない．このため，4.2.3 項と同様の手法では取得できない．そこで，3.6.3 項で述べた（手法 B）を利用する．

（手法 B）は，データ構造の解析が必要なため，以降では 4.1 節の実現環境におけるデータ構造について述べる．本稿において，ゲスト OS として用いた Debian 7.1.0-i386 (Linux 3.2.46) におけるプロセスとファイルを管理する構

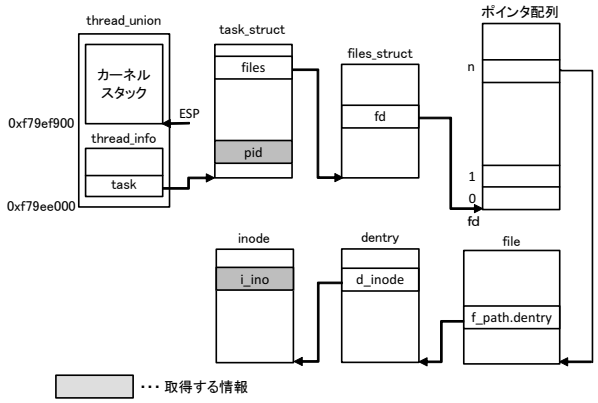

図 3 プロセスとファイルを管理する構造体

造体を図 **3** に示す．

プロセス情報は，プロセス制御ブロックである task_struct 構造体に格納されている．また，task_struct 構造体の開始アドレスは，thread_info 構造体の task メンバから参照されている．thread_info 構造体の先頭アドレスをカーネルスタックのスタックポインタを利用して取得し，図 3 に示したように構造体の各メンバを辿ることにより，PID と inode 番号を取得する．なお，inode 番号の取得には，システムコールをフックした際に取得した fd を利用する．

5. 関連研究

SVFS[8] は VMM 上でユーザが作業を行う Normal VM，管理用の Admin VM，およびファイルサーバの役割を持つ DVM の 3 つの VM を起動する．システムファイルや機密情報ファイルは Admin VM からのみ編集できる．これにより，Normal VM が攻撃者に乗っ取られた場合でも，システムファイルや機密情報を保護できる．上記の研究は，VMM 上で複数の OS を起動させ，ユーザが作業を行う OS とは別の OS にファイルアクセスを制御する機能を実現することにより，機能を保護する．一方，VMM における拡散追跡機能は，VMM 内に機密情報を追跡する機能を実現することにより機能を保護する．

VOFS[9] は，VMM 上でユーザが作業を行う Primary Guest VM と機密情報を管理する SVFS VM を動作させる．Primary Guest OS で機密情報を閲覧する際，SVFS VM に要求を出し，Content Server に保存されている機密情報を提示する．このとき，ディスクとネットワークへの出力を無効化することにより，機密情報の漏えいを防ぐ．文献 [10] は，VMM を用いて，機密情報閲覧作成環境を実現している．機密情報閲覧作成環境において，機密情報は機密情報保存サーバに保存する．クライアント側では，VMM 上でユーザが作業を行う作業 OS と作業 OS のファイルアクセスを制御する管理 OS の 2 つの OS を動作させる．そして，ユーザが機密情報の閲覧や作成を行う際に，管理 OS により，作業 OS のディスクとネットワークへの出力を無効化することにより，機密情報の漏えいを防ぐ．これらの研究は，ディスクとネットワークへの出力を無効化することにより，機密情報の外部への漏えいを防止する．一方，VMM における拡散追跡機能はシステムコールの発行を監視し，機密情報を計算機外部へ送信するシステムコールを停止することにより，機密情報の外部への漏えいを防止する．

また，TaintEraser[11] は，Dynamic Taint Analysis により，機密情報の拡散を追跡する．Dynamic Taint Analysis は，まず，機密情報に Taint を付加する．その後，Taint の付加された情報がメモリ内の他の場所に書き出された際に，書き出し先にも Taint を付加する．これにより，Taint が付加された情報を機密情報として把握することができる．同様の技術を用いた研究に TaintDroid[12] や文献 [13] がある．TaintDroid は，スマートフォンに実現されており，端末内の機密情報の拡散を追跡する．また，機密情報が端末外に漏えいした場合，利用者に通知する．文献 [13] は，Taint を用いたマルウェアの解析環境を実現している．この手法は，仮想環境で実現されており，ゲスト OS 上でマルウェアを解析する．シャドウメモリやシャドウディスクを用意し，シャドウ領域に Taint を付加することで，ゲスト OS は Taint の存在を意識することなく動作する．これらの研究は Taint を付加するためにシャドウ領域を用意する．このため，メモリやディスクの使用量が増大する．一方，提案手法は，シャドウ領域に相当する必要がないため，メモリやディスクの使用量を抑えた上で，機密情報の拡散を追跡できる．しかし，VMM における拡散追跡機能は，システムコールで取得できる情報で追跡するため，追跡の粒度はこれらの手法よりも劣る．

6. おわりに

KVM を用いた機密情報の拡散追跡機能の設計を述べた．VMM における拡散追跡機能は，KVM 内に機密情報の拡散追跡機能を実現することにより，ゲスト OS 内における機密情報の拡散を追跡する．VMM における拡散追跡機能は，ゲスト OS におけるシステムコールの発行を検知し，処理をフックする．その後，システムコールを発行したプロセス，プロセスが扱うファイル，およびプロセスの通信先を特定することにより，機密情報の拡散を追跡する．また，システムコール処理からの復帰を検知し，システムコールの戻り値を取得することで，より正確に機密情報の拡散を追跡する．

また，KVM 内に機密情報の拡散追跡機能を実現するため，導入対象の OS のソースコードを修正する必要はない．これにより，機密情報の拡散追跡機能をより多くの環境に導入可能になる．

さらに，機密情報の拡散追跡機能を OS から隔離できるため，既存機能の問題であった機密情報を搾取しようとす

る攻撃者や悪意のある計算機の利用者による機密情報の拡散追跡機能への攻撃をより困難にする．

今後の課題として，機密情報の拡散に関係するすべてのシステムコールをフックすることによる機密情報の漏れのない追跡，および VMM における拡散追跡機能の導入による性能低下の抑制と評価がある．

参考文献

[1] 日本ネットワークセキュリティ協会：2012 年個人情報漏えいインシデント調査結果 Ver.1.3（オンライン），日本ネットワークセキュリティ協会（オンライン），入手先 ⟨http://www.jnsa.org/result/incident/2012.html⟩（参照 2014-01-10）．

[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] RBB TODAY: 正規セキュリティソフトが、攻撃ツールに変身する可能性……カスペルスキーが問題点を指摘，RBB TODAY（オンライン），入手先 ⟨http://www.rbbtoday.com/article/2014/02/25/117208.html⟩（参照 2014-05-14）．

[4] Chen, P. M. and Noble, B. D.: When Virtual Is Better Than Real, *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, IEEE Computer Society, pp. 133–138 (2001).

[5] KVM: Main Page, KVM (online), available from ⟨http://www.linux-kvm.org/page/Main_Page⟩ (accessed 2014-01-09).

[6] Dinaburg, A., Royal, P., Sharif, M. and Lee, W.: Ether: Malware Analysis via Hardware Virtualization Extensions, *Proceedings of the 15th ACM Conference on Computer and Communications Security*, ACM, pp. 51–62 (2008).

[7] Intel: Intel 64 and IA-32 Architectures Software Developer's Manual Combined Volumes:1, 2A, 2B, 2C, 3A,3B, and 3C, Intel (online), available from ⟨http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf⟩ (accessed 2014-01-10).

[8] Zhao, X., Borders, K. and Prakash, A.: Towards protecting sensitive files in a compromised system, In Proc. Third IEEE International Security in Storage Workshop (SISW'05)，IEEE Computer Society, pp. 21–28 (2005).

[9] Borders, K., Zhao, X. and Prakash, A.: Securing Sensitive Content in a View-only File System, *Proceedings of the ACM Workshop on Digital Rights Management*, ACM, pp. 27–36 (2006).

[10] 栗本裕司，齋藤彰一，松尾啓志：ディスク擬似書き込みと仮想マシンモニタによる機密情報閲覧作成環境の実現，情報処理学会研究報告，Vol. 2010-CSEC-48, No. 54, pp. 1–8 (2010).

[11] Zhu, D. Y., Jung, J., Song, D., Kohno, T. and Wetherall, D.: TaintEraser: Protecting Sensitive Data Leaks Using Application-level Taint Tracking, Vol. 45, No. 1, pp. 142–154 (2011).

[12] Enck, W., Gilbert, P., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P. and Sheth, A. N.: TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones, *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, USENIX Association, pp. 1–6 (2010).

[13] 川古谷裕平，塩治榮太朗，岩村　誠，針生剛男：テイント伝搬に基づく解析対象コードの追跡方法，コンピュータセキュリティシンポジウム 2012 論文集，Vol. 2012, No. 3, pp. 1–8 (2012).