プロセス間通信による機密情報拡散を KVM 上で追跡する機能の設計

藤井 翔太　　佐藤 将也　　山内 利宏　　谷口 秀夫

岡山大学大学院自然科学研究科
700-8530 岡山県岡山市北区津島中 3-1-1
fujii@swlab.cs.okayama-u.ac.jp {sato, yamauchi, tani}@cs.okayama-u.ac.jp

あらまし 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た．また，機密情報の拡散経路のうち，プロセス間通信を除くファイル操作とプロセス生成による機密情報の拡散を KVM から追跡する機能を実現した．本稿では，プロセス間通信の 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について述べる．また，プロトタイプを実装し，評価した結果を述べる．

Design of Function for Tracing Diffusion of Classified Information for IPC on KVM

Shota Fujii　　Masaya Sato　　Toshihiro Yamauchi　　Hideo Taniguchi

Graduate School of Natural Science and Technology, Okayama University
3-1-1, Tsushima-naka, Kita-ku, Okayama, 700-8530, JAPAN

Abstract We have proposed a function for tracing the diffusion of classified information in a guest OS using a VMM. In addition, we have implemented a proposed function for file operations and process creation excluding inter-process communication with a KVM in previous work. In this paper, we describe the design and implementation of the proposed function for inter-process communication with a KVM. Further, we report the result of evaluations.

1　はじめに

計算機の普及とともに，個人情報をはじめとした機密情報を計算機で扱う場面が増加している．これに伴い，機密情報の計算機外部への漏えい事例が増加し，問題となっている．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 67%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できることが重要である．また，機密情報を狙うサイバー攻撃は巧妙化しており，完全に防ぐことは困難になっている．そこで，機密情報が計算機外部に送信されようとしていることを検知し，ユーザの損失を可能な限り少なくすることが重要となっている [2].

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を実現した [3]．また，他にも機密情報の拡散追跡機能を利用して拡散経路を可視化する機能 [4] や複数計算機間の機密情報拡散を追跡する機能 [5] を実現した．これらの機能は，機密情報の利用状況の把握や漏えいの防止に効果がある．

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 と呼ぶ）内で動作するため，その存在を検知され，無効化される可能性がある．無効化されてしまうと，機密情報の漏えいを検知できず，被害が拡大する可能性がある．また，導入の際，対象の OS （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，導入環境が限定される問題がある．

機密情報の拡散追跡機能と同様に，機密情報の保護を目的とした研究は多数存在する [6],[7]．しかし，いずれも，機密情報の拡散追跡機能と同様に，無効化される可能性や導入環境が限定的である問題がある．これらの問題への対処として，OS 外部から機密情報を保護する手法 [8],[9] が研究されており，OS 外部に機構を実現することの有効性が示されている．しかし，これらの手法は情報漏えいの防止のみを目的としており，機密情報の拡散を追跡していないため，機密情報の利用状況を把握することは難しい．

そこで，我々は，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）を設計した [10]．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．また，VMM は，OS よりも攻撃が困難であるため，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．我々はこれまでに，3 つの情報拡散経路のうち，ファイル操作と子プロセス生成について実現と評価を行った [11]．しかし，プロセス間通信の追跡機能については実現できていない．

そこで，本稿では Linux におけるプロセス間通信による機密情報拡散のうち，ソケット通信による機密情報の拡散を Kernel-based Virtual Machine（以降，KVM と呼ぶ）[12] 上で追跡する機能の設計と実現方式を述べる．また，実現方式に基づいて実装を行い，ソケット通信の VMM による追跡可能性と性能を評価した結果を報告する．

2 仮想計算機モニタにおける機密情報の拡散追跡機能

2.1 機密情報の拡散経路

文献 [3] で実現した機密情報の拡散追跡機能（以降，既存機能と呼ぶ）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する．以降，これらのファイルとプロセスを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，さらに他のファイルやプロセスなどにその内容を伝えることにより起こる．以下にプロセスが情報を伝達する 3 つの処理を

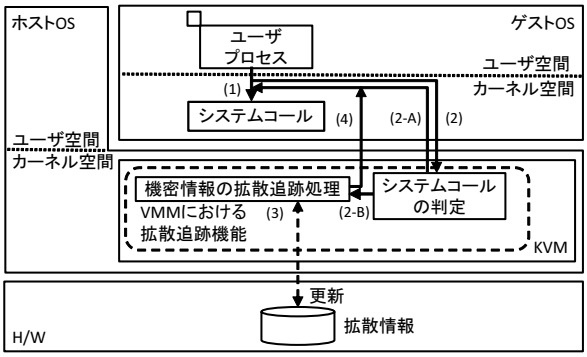


図 1 VMM における拡散追跡機能の全体像

示す．これらの処理を監視し，機密情報の拡散を追跡する．

(1) ファイル操作
(2) 子プロセス生成
(3) プロセス間通信

2.2 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 により実現する．具体的には，ゲスト OS 上の機密情報を有する可能性があるファイルとプロセスを VMM から管理する．また，ゲスト OS 上で実行される 2.1 節で述べた 3 つの処理を VMM から追跡し，機密情報の伝搬状況やその経路を把握する．利用者は，VMM に保管された管理対象ファイルリストによって，機密情報の所在を把握できる．さらに，機密情報の拡散を検知した際には，書き出し先のパス名，拡散したファイルの inode 番号，書き出しに用いたコマンド名，およびプロセス ID（以降，PID と呼ぶ）が記録される．このため，機密情報が漏えいした場合でも，その情報から利用者は情報漏えいをより早く検知でき，被害を抑制することができる．

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，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 プロセス間通信による機密情報拡散の追跡機能の設計

3.1 追跡対象

文献 [11] では，(1) ファイル操作と (2) 子プロセス生成を追跡する手法を述べた．そこで，本稿では，残る (3) プロセス間通信を VMM における拡散追跡機能を用いて追跡する手法を述べる．

プロセス間通信には，パイプ，FIFO，メッセージキュー，共有メモリ，およびソケットなど，多くの手法がある．なかでも，ソケットを用いたプロセス間通信は，ローカルプロセス間通信だけでなく，リモートプロセス間通信にも利用される．リモートプロセス間通信を行った場合，機密情報が計算機外部に漏えいする可能性があり，ローカルプロセス間通信に比べ，追跡の重要性が高い．そこで，本稿では，ソケット通信の追跡手法について述べる．

3.2 課題

VMM における拡散追跡機能を用いて，ソケット通信による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) VMM によるシステムコールのフック
(課題 2) ソケット通信による機密情報の拡散と漏えい契機の検知
(課題 3) 追跡に必要な情報の取得と追跡

VMM からソケット通信を追跡するには，VMM からゲスト OS 上で発行されるソケット通信のシステムコールの発行を検知し，処理をフックする必要がある．また，ソケット通信による機密情報の拡散や漏えい契機を漏れなく検知する必要がある．さらに，ソケット通信では，ソケットという媒体を介して通信を行うことや計算機内部だけでなく計算機外部とも通信する可能性があることから，ファイル操作や子プロセス生成の追跡に比べて追跡処理が複雑になり，それらの追跡とは同様の手法を取ることができない．そこで，ソケット通信を追跡するために必要な情報を取得し，適切な追跡処理を行う必要がある．このとき，プロセス間の通信経路となるソケットを管理対象として管理し，伝播の契機を検知する必要がある．

表 1 ソケット通信に使用されるシステムコール

役割	システムコール名	拡散可能性	漏えい可能性	追跡必要性
ソケット生成	socket	×	×	×
ソケット登録	bind	×	×	×
接続準備	listen	×	×	×
接続待機	accept	×	×	×
接続要求	connect	×	×	×
送信	sendto	○	○	○
	sendmsg	○	○	○
	write	○	○	○
	sendfile	○	○	○
受信	recvfrom	○	×	○
	recvmsg	○	×	○
	read	○	×	○
ソケット解放	shutdown	×	×	○
	close	×	×	○

（課題 1）については，文献 [10],[11] で設計・実現済みである．（課題 2）については，ソケット通信による機密情報の拡散や漏えいを引き起こすシステムコールを追跡対象とすることにより，対処する．また，（課題 3）については，ローカルプロセス間通信とリモートプロセス間通信の 2 通りの場合があるため，それぞれへの対処方法を述べる．

3.3 ソケット通信による機密情報の拡散と漏えい契機の検知

ソケット通信に使用されるシステムコール名と各システムコールの役割，情報拡散の可能性，情報漏えいの可能性，および追跡の必要性を表 1 に示す．なお，表に示すシステムコールは，Linux 3.6.10 の場合のものであり，導入対象 OS のバージョンによって異なる．

表 1 のソケット生成から接続要求までの処理は，データがやり取りされず，機密情報の拡散や漏えいは発生しない．このため，ソケット生成から接続要求までの処理を行うシステムコールは追跡対象としない．

機密情報の拡散は，送信システムコールや受信システムコールによって実際に通信が行われた際に発生するため，両システムコール（以降，送受信システムコールと呼ぶ）を追跡対象とする．また，機密情報の漏えいは，機密情報が送信システムコールによって計算機外部に送信される際に発生するため，送信システムコールの処理をフックした際には，情

報漏えいの可能性も検証する．

さらに，解放されたソケットは通信に利用されることはないため，ソケット解放システムコールを追跡し，ソケットが解放される際に，管理対象から除外する．

3.4 追跡に必要な情報の取得と追跡

3.4.1 ローカルプロセス間通信による機密情報拡散の追跡

ソケット通信によるローカルプロセス間通信では，送受信システムコールをフックし，通信媒体であるソケットを特定する．このとき，ソケットが機密情報を有するか否かを管理することにより，ローカルプロセス間通信による機密情報の拡散を追跡する．

3.4.2 リモートプロセス間通信による機密情報漏えいの検知

リモートプロセス間通信では，リモートプロセス間通信を行う送信システムコールをフックし，リモートプロセス間通信による機密情報の漏えいを検知する．このとき，通信先の計算機とプロセスを特定するために，通信先の IP アドレスとポート番号を取得する．

3.5 ソケット通信追跡の処理流れ

本章で述べた内容をもとに，VMM からソケット通信を追跡する際の処理流れを以下に示す．

(1) 送信システムコールを VMM からフック
(2) 送信システムコールを発行したプロセスが管理対象であれば，以下の処理を実行
  (A)ローカルプロセス間通信の場合，送信先（受信側）ソケットを管理対象に追加
  (B)リモートプロセス間通信の場合，情報漏えいの可能性として検知し，その旨をログに出力
(3) 送信システムコールの処理をゲスト OS に返却し，ゲスト OS はデータの送信処理を実行
(4) 受信システムコールを VMM からフック
(5) 受信側プロセスのソケットが管理対象であれば，受信システムコールを発行したプロセスを管理対象に追加
(6) 受信システムコールの処理をゲスト OS に返却し，ゲスト OS はデータの受信処理を実行

4 実現方式

4.1 実現環境

ゲスト OS，ホスト OS ともに Linux 3.6.10（64bit），VMM に KVM を利用する．また，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には，SYSCALL/SYSRET を利用することを前提とする．さらに，Intel 社の仮想化支援機能である VT-x の利用を前提とする．

4.2 ソケット通信追跡の要件と課題

3 章における設計から，VMM からソケット通信による機密情報の拡散を追跡し，漏えいを検知するための要件を以下に示す．

(要件 1) ローカルプロセス間の機密情報拡散の追跡
(要件 2) リモートプロセス間の機密情報漏えいの検知
(要件 3) VMM から取得可能な情報のみによる機密情報拡散の追跡と漏えいの検知

ソケット通信の追跡には，3.4 節で述べたローカルプロセス間の機密情報拡散の追跡とリモートプロセス間の機密情報漏えいの検知が必要である．また，VMM における拡散追跡機能は，VMM 内に実現されているため，上記の追跡や検知を VMM から取得可能な情報のみを用いて行う必要がある．

Linux において，ソケット通信によるローカルプロセス間通信には，UNIX ドメインによる通信と INET ドメインによるループバックアドレスや自ホストアドレスへの通信がある．また，ソケット通信によるリモートプロセス間通信には，INET ドメインによる外部計算機のアドレスへの通信がある．

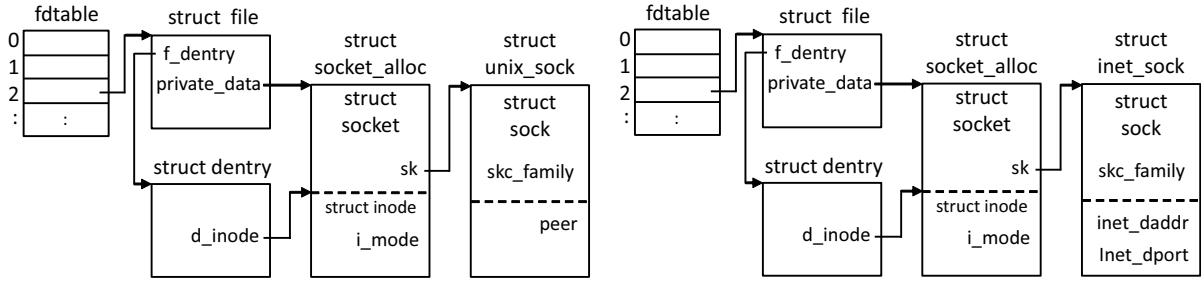
以上より，（要件 1）～（要件 3）を満足し，機能を実現するための課題として，以下の 2 つが挙げられる．

(実現課題 1) VMM からの UNIX ドメインによるソケット通信の追跡
(実現課題 2) VMM からの INET ドメインによるソケット通信の追跡

以降の節で上記の実現課題を解決する方法を述べる．

4.3 VMM からの UNIX ドメインにおけるソケット通信の追跡

3.4.1 項で述べたように，ローカルプロセス間通信を追跡するために，送受信システムコールの対象ソケットが機密情報を有するか否かを管理する．ここ



(A) UNIX ドメイン　　(B) INET ドメイン

図 2　ソケットに関するデータ構造間の関係

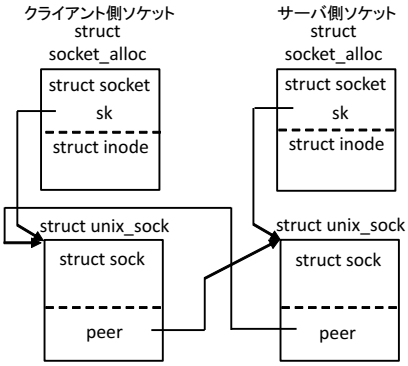


図 3　接続されたソケット間のデータ構造の関係（UNIX ドメイン）

で，UNIX ドメインにおけるソケットに関するデータ構造間の関係を図 2-(A) に示す．ソケットの実体は，`socket` 構造体と `unix_sock` 構造体であり，図からわかるように，ファイルディスクリプタ（以降，fd と呼ぶ）を用いてデータ構造をたどることにより，取得できる．また，送受信システムコールを VMM からフックし，引数の fd を取得することにより，システムコールが扱うソケットを特定する．

また，機密情報は，送信システムコールにより送信側のソケットを介して受信側のソケットへ拡散する．このため，送信システムコールの対象ソケットを fd から特定した後，対象ソケットの通信先ソケットも特定する必要がある．両ソケット間の接続は，サーバ側の accept システムコールで接続を待つソケットに対して，クライアント側で connect システムコールにより接続要求を出すことにより確立される．接続後のソケット間のデータ構造の関係を図 3 に示す．図 3 より，ソケットの通信先は，`unix_sock` 構造体の `peer` メンバから特定できることが分かる．そこで，送信システムコールフック時に特定した送信側ソケットのデータ構造をたどり，`unix_sock` 構造体の `peer` メンバから受信側ソケットを特定する．

さらに，解放されたソケットを管理対象から除外するために，ソケット解放システムコールをフックし，ソケットが解放される際，管理対象から除外する．

4.4　VMM からの INET のドメインにおけるソケット通信の追跡

INET ドメインの場合，UNIX ドメインとは異なり，ソケットが同一の計算機上に存在するとは限らないため，通信先を特定するために，送信先（受信側）ソケットの IP アドレスとポート番号を利用する．ここで，INET ドメインにおけるソケットに関するデータ構造間の関係を図 2-(B) に示す．INET ドメインでのソケットの実体は，`socket` 構造体と `inet_sock` 構造体であり，UNIX ドメインのソケットと同様に，fd を用いてデータ構造をたどることにより，取得できる．また，通信先 IP アドレスとポート番号は，それぞれ `inet_sock` 構造体の `inet_daddr` メンバと `inet_dport` メンバに格納されている．

そこで，送受信システムコールをフックし，ソケットのデータ構造から各メンバを取得する．このとき，通信先 IP アドレスがループバックアドレスか自ホストのアドレスの際に，送信側からみた送信先ソケットのポート番号と受信側からみた受信側ソケットのポート番号が一致した場合，両者が通信したと判断し，情報拡散を追跡する．また，送信システムコールによるデータの通信先がループバックアドレスや自ホストのアドレス以外の場合は，リモートへの通信だと判断し，情報漏えいの可能性があるとして検知する．

5　評価

5.1　評価項目

以下に，評価項目とそれぞれの目的を述べる．

(1) 追跡可能性
(2) システムコールのオーバヘッド

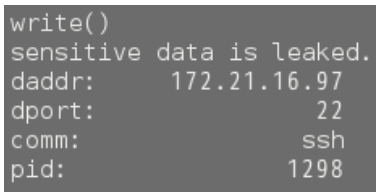


図 4 VMM における拡散追跡機能が（事例 2）の機密情報の漏えいを検知した際に発行したログ

(3) マイクロベンチマーク

ソケット通信ではローカルプロセス間通信による機密情報の拡散とリモートプロセス間通信による機密情報の漏えいが発生する可能性があり，この両方を検知できることが VMM における拡散追跡機能に要求される．そこで，この 2 つの事例を実際にゲスト OS 上で行い，VMM から追跡可能か否か検証する．また，当該システムコールに生じるオーバヘッドの測定やベンチマークによる性能測定を実施し，どの程度性能が低下するかを検証する．

なお，評価に用いたマシンは，CPU が Intel Core i5-3470 (3.2 GHz, 4 CPUs)，メモリが 4,096 MB である．また，ゲストには，仮想 CPU 1 つと 1,024 MB のメモリを割り当てた．評価では，Hyper-Threading と EPT を無効にしている．

5.2 追跡可能性

5.2.1 評価方法

以下に示す 2 つの事例を用いて VMM からのソケット通信による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) UNIX ドメインソケットによるローカルプロセス間通信
UNIX ドメインソケットによるクライアント・サーバプログラムを用いて，管理対象のクライアントと非管理対象のサーバ間でプロセス間通信を行う．

(事例 2) 計算機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計算機外部へ送信する．

5.2.2 評価結果

ローカルプロセス間通信においては，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，別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，そのプロセスにも機密情報が拡散する．（事例 1）を行った際，ソケットに機密情報を送信する sendto() をフックした VMM における拡散追跡機能は，ソケットへの機密情報の拡散を検知し，ソケットを新たな管理対象に加えた．また，ソケットから機密情報を受信する recvfrom() をフックした際に，ソケットからプロセスへの機密情報の拡散を検知し，機密情報を受信したプロセスを新たな管理対象に加えた．

また，リモートプロセス間通信においては，管理対象プロセスが機密情報をリモート計算機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，scp コマンド中で機密情報を計算機外部に送信する write() をフックした際に，機密情報の漏えいを検知した．また，VMM における拡散追跡機能が機密情報の漏えいを検知した際に発行したログを図 4 に示す．図に示すように，送信先の IP アドレス（daddr），通信に利用されたポート番号（dport）とコマンド（comm），および PID（pid）を記録しており，利用者は記録から漏えい原因の特定が可能である．

以上より，VMM における拡散追跡機能は，ローカルプロセス間通信による機密情報拡散の追跡とリモートプロセス間通信による機密情報漏えいの検知の 2 つの要求を満たしていることが確認できた．

5.3 システムコールのオーバヘッド

5.3.1 評価方法

VMM における拡散追跡機能によって生じるオーバヘッドを測定するために，VMM における拡散追跡機能の導入前と導入後のゲスト OS において性能を測定した．また，機能導入後において，プロセスが管理対象の場合とそうでない場合で測定結果に差が出るかを検証するために，両方の場合で測定した．このとき，追跡対象である UNIX ドメインの sendto() と recvfrom() を測定対象とし，各システムコールに生じるオーバヘッドを測定した．また，比較のために，非追跡対象のシステムコールである getpid() も測定対象とした．

5.3.2 評価結果

測定結果を表 2 に示す．追跡対象である sendto() と recvfrom() にはそれぞれ，非管理対象の場合は実時間で 23.76μs と 44.77μs のオーバヘッド，相対性能は 283.83%と 460.36%，管理対象の場合は実時間で 34.18μs と 49.58μs のオーバヘッド，相対性能は 364.48%と 499.10%と比較的大きい値になっている．機能導入後が機能導入前に比べて処理時間が長大化しているのは，プロセスやソケットが管理対象

表 2 VMM における拡散追跡機能におけるシステムコールのオーバヘッド（μs）

システムコール名	機能導入前	機能導入後（相対性能）		オーバヘッド	
		非管理対象の場合	管理対象の場合	非管理対象の場合	管理対象の場合
sendto	12.92	36.68 (283.83%)	47.10 (364.48%)	23.76	34.18
recvfrom	12.42	57.19 (460.36%)	62.00 (499.10%)	44.77	49.58
getpid	0.015	0.016 (106.86%)	-	0.0011	-

表 3 プロセス間通信のレイテンシ（μs）

項目	機能導入前	機能導入後（相対性能）		オーバヘッド
slct TCP	2.08	6.82	(327.88%)	4.74
AF UNIX	7.24	39.57	(546.55%)	32.33
UDP	13.33	63.23	(474.34%)	49.90
TCP	16.77	60.50	(360.76%)	43.73
TCP conn	80.33	133.00	(165.57%)	52.67

か否かを判定する処理の際，それぞれを管理するリストを走査するためである．また，管理対象プロセスや管理対象ソケットの数に比例して，機能導入後のオーバヘッドが増大すると考えられる．さらに，機能導入後で送信元のプロセスが管理対象の場合，sendto() のフック時には送信先のソケットを管理対象に加える処理，recvfrom() のフック時には受信したプロセスを管理対象に加える処理を行う．この処理のために，それぞれ非管理対象の場合に比べて処理時間が長大化している．

一方で，非追跡対象システムコールである getpid() には，0.0011μs と比較的小さいオーバヘッドしか生じておらず，相対性能で見ても，106.86%である．非追跡対象システムコールの場合，システムコールをフック後，追跡対象システムコールか否かを確認するだけであるため，性能への影響は小さい．

5.4 マイクロベンチマーク

5.4.1 評価方法

マイクロベンチマークである LMbench[13] を用いて，プロセス間通信のレイテンシを測定した．このとき，システムコールと同様に，VMM における拡散追跡機能の導入前と導入後のゲスト OS において測定を行った．

5.4.2 評価結果

LMbench を用いて測定したプロセス間通信のレイテンシを表 3 に示す．slct TCP，AF UNIX，UDP，TCP，および TCP conn は，それぞれソケット read･write の合計レイテンシ，ソケットによるプロセス間通信待ち時間，UDP/IP 経由のプロセス間通信待ち時間，TCP/IP 経由のプロセス間通信待ち時間，およびソケットのコネクション確立までの時間である．

送受信処理を行う slct TCP，AF UNIX，UDP，および TCP には，約 5∼50μs のオーバヘッドが生じており，相対性能も 327.88∼546.55%となっている．これは，5.3 節の評価と同様に，プロセスやソケットが管理対象か否かを判定する処理により生じていると考えられる．また，TCP conn には，約 53μs と最も大きいオーバヘッドが生じている．これは，ソケットのコネクション確立までに，多くのシステムコールが発行され，そのたびに処理がゲスト OS から VMM に遷移することが原因だと考えられる．ただし，この間に発行されるシステムコールには機密情報拡散の追跡処理や漏えいの検知処理は行わないため，相対性能は 166.57%とオーバヘッドの増加割合は他に比べて小さい．

6 関連研究

機密情報漏えいの防止を目的とした研究に，TightLip[6] がある．TightLip は，機密情報を読み込んだプロセスがネットワークを通じて書き出しを行おうとした際に，機密情報を含むファイル内容ではなく，機密情報を含まないシャドウファイルの内容を書き出すため，機密情報の漏えいを防止できる．Aquifer[7] は，ポリシを用いて，機密データ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．Filesafe[8] は，VMM を用いてゲスト OS 上のファイルを保護する．ユーザは，機密情報を有するファイルに対して read/write の可否をあらかじめ設定しておく．その後，ゲスト OS で発行された read/write 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，意図しない情報の漏えいを防止する．SVFS[9] 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．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，機密情報を保護できる．また，VOFS[14] は，SVFS を用いて，機密情報の閲覧のみを利用者に許可するシステムである．

TightLip, Aquifer, SVFS, および VOFS は，OS の構造を変更する必要があり，導入可能な環境に制限がある．それに対し，提案手法は，ゲスト OS の構造を変更する必要はないため，より多くの環境に導入可能である．また，FileSafe は各ファイルに逐一ポリシ設定を行う必要があり，ポリシに漏れがあった場合，ユーザの意図しない情報漏えいが発生しうる可能性がある．一方，提案手法は機密情報が拡散する度に自動で追跡を行うため，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．

7 おわりに

VMM における拡散追跡機能を用いてプロセス間通信の 1 つであるソケット通信を追跡する手法を述べた．ローカルプロセス間通信の場合は，プロセスとソケットが機密情報を有するかを管理することにより，機密情報の拡散を追跡する．また，リモートプロセス間通信の場合は，管理対象プロセスが機密情報を計算機外部に送信することを検知する．これによって，機密情報の漏えいを検知するとともに，送信先の IP アドレス，通信に利用されたポート番号とコマンド，および PID を記録することにより，漏えい原因の特定を可能にする．

また，VMM に KVM，ゲスト OS に Linux を用いて実装し，評価を行った．事例を用いた評価では，ローカルプロセス間通信による機密情報拡散の追跡とリモートプロセス間通信による機密情報漏えいの検知の両方を VMM から可能であることを確認した．性能評価では，追跡対象のシステムコールには，実時間で 23.76μs ～ 49.58μs，相対性能で 283.83% ～ 499.10% と比較的大きいオーバヘッドが生じることと非追跡対象システムコールでは，実時間で 0.0011μs，相対性能で 106.86%と小さいオーバヘッドであることを示した．

さらに，パイプ，FIFO，メッセージキュー，および共有メモリなどのプロセス間通信は，ソケット通信と同様に，パイプ，メッセージキューおよび共有メモリといった通信媒体を介してプロセス間での通信を実現している．このため，ソケット通信以外のプロセス間通信でも，それぞれのプロセス間通信において通信を行うシステムコールをフックし，通信媒体が機密情報を有するか否かを管理する提案手法を適用することで，ソケットと同様に追跡可能である．

残された課題として，ソケット通信以外によるプロセス間通信の追跡機能の実現がある．

参考文献

[1] 日本ネットワークセキュリティ協会：2013 年個人情報漏えいインシデントに関する調査報告書 ～ 個人情報漏えい編 ～，http://www.jnsa.org/result/incident/.

[2] thegurdian: Antivirus software is dead, says security expert at Symantec, http://www.theguardian.com/technology/2014/may/06/antivirus-software-fails-catch-attacks-security-expert-symantec.

[3] 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）.

[4] Nomura, Y., Hakomori, S., Yokoyama, K., and Taniguchi, H.: Tracing the Diffusion of Classified Information Triggered by File Open System Call, *Proceedings of 4th Int. Conf. on Computing, Communications and Control Technologies (CCCT 2006)*, pp.312–317 (2006).

[5] 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).

[6] Yumerefendi, A. R., Mickle, B. and Cox, L. P.: TightLip: Keeping Applications from Spilling the Beans, *Proceedings of the 4th USENIX Conference on Networked Systems Design and Implementation (NSDI 2007)*, pp.159–172 (2007).

[7] 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).

[8] 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).

[9] Zhao, X., Borders, K., and Prakash, A.: Towards protecting sensitive files in a compromised system, *In Proceedings of Third IEEE International Security in Storage Workshop (SISW05)*, pp.21–28 (2005).

[10] 藤井 翔太，山内 利宏，谷口 秀夫：KVM における機密情報の拡散追跡機能，情報処理学会研究報告，Vol.2014-CSEC-66，No.28，pp.1–7（2014）.

[11] 藤井翔太，山内利宏，谷口秀夫：ファイル操作による機密情報拡散を KVM 上で追跡する機能の評価，コンピュータセキュリティシンポジウム 2014 論文集，pp.751–758 (2014).

[12] KVM: Main_Page, http://www.linux-kvm.org/page/Main_Page.

[13] McVoy, L. and Staelin, C.: lmbench: Portable tools for performance analysis, *Proceedings of the USENIX Annual Technical Conference*, pp.279–294（1996）.

[14] 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*, pp.27–36 (2006).