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

藤井 翔太　　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 {yamauchi, tani}@cs.okayama-u.ac.jp

あらまし 近年，機密情報が漏えいする事例が増加している．この問題へ対処するために，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．また，仮想計算機モニタにおける機密情報の拡散追跡機能を設計した．本稿では，ファイル操作とプロセス生成による機密情報の拡散追跡機能の KVM 内での実装方式について述べる．また，実装した機能の改変コード量とオーバヘッド量を報告する．

Evaluation of Tracing Classified Information Diffusion for File Operations on KVM

Shota Fujii　　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 Recently, the cases of classified information leakage are increasing. To address this problem, we realized the function for tracing classified information diffusion within OS. The function traces classified information diffusion and manages resources which have classified information. In addition, we designed the function for tracing a guest OS's classified information diffusion using virtual machine monitor. This paper describes the implementation of the function for file operations and process creation with the KVM. Further, this paper reports the amount of modified source codes and overheads.

1 はじめに

個人情報の価値が上昇するとともに，情報漏えいの防止への需要が高まっている．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 79%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．一方で，機密情報の窃取を目的とした事件も発生しており，機密情報が漏えいしたことが感知できない場合，さらなる被害につながる可能性もある．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を実現した [2].

また，文献 [3] において，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）の設計を述べた．機密情報の拡散追跡機能を VMM 内に実現することにより，導入対象 OS のソースコードの修正が不要，機密情報の拡散追跡機能への攻撃が困難，および，カーネルのバージョンの変更に対応可能といった効果が期待できる．

本稿では，文献 [3] の設計に基づき，ファイル操作とプロセス生成による機密情報の拡散追跡機能の KVM（Kernel-based Virtual Machine [4]）内での

実現方式を述べる．また，実装した機能の改変コード量とオーバヘッド量を報告する．

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 [2] で実現されている機密情報の拡散追跡機能（以降，既存機能）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する．以降，これらのファイルや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 つの処理を示す．

(1) ファイル操作
(2) プロセス間通信
(3) 子プロセス生成

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

文献 [3] の VMM における拡散追跡機能は，既存機能と同等の追跡機能を VMM 内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 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．

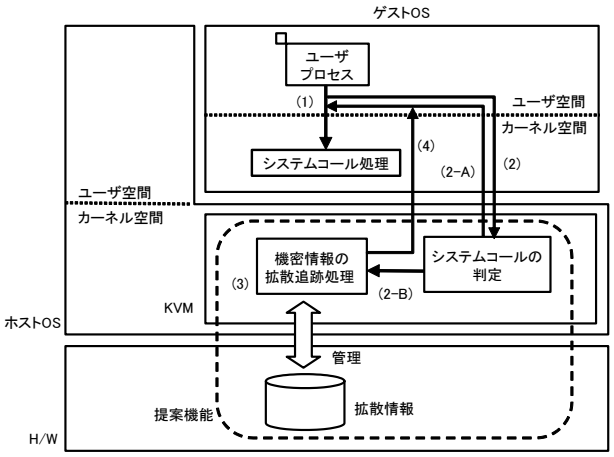

図 1 VMM における拡散追跡機能の全体図

2.3 課題

VMM 内に機密情報の拡散追跡機能を実現する際の課題に，以下の 2 点がある．

(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．

2.4 仮想計算機モニタによるシステムコールの情報の取得

2.4.1 システムコール入口のフック

ゲスト OS におけるシステムコールの発行を VMM により検知するために，システムコールの入口（sysenter）をフックする．sysenter 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．sysenter のフックは，ゲスト OS の利用する `sysenter_eip_msr` を書き換えることで実現する．

2.4.2 システムコール出口のフック

各システムコールは，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を返却する．VMM における拡散追跡機能は，機密情報を追跡するために，プロセスが扱うファイルの情報を VMM で把握する

必要がある．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．

そこで，VMM における拡散追跡機能では，システムコールの出口（sysexit）もフックする．sysexit 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．sysexit のフックは，ハードウェアブレークポイントにより，実現する．

2.4.3 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，通信先のファイルやプロセスを特定できる．さらに，2.4.2 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．

2.5 仮想計算機モニタによる OS の情報の取得

2.3 節で述べたように，機密情報の拡散追跡機能は，ファイル情報やプロセス情報といった OS の情報を利用し，機密情報の拡散を追跡する．このとき，VMM から OS の情報を取得するためには，VM と VMM 間のセマンティックギャップ [5] を解決する必要がある．セマンティックギャップの解決には，VMM 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 の情報を取得する手法を利用する．

2.6 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
(効果 2) 機密情報の拡散追跡機能への攻撃が困難
(効果 3) カーネルのバージョンの変更に対応可能

3 実現方式

3.1 実現環境

本章では，実現環境として，VMM に KVM，ゲスト OS に Linux（Linux カーネル 3.2.46）の利用を想定する．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．このため，ゲスト OS におけるシステムコールの発行には，sysenter/sysexit を利用することを前提とする．また，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，Intel 社の仮想化支援機能である VT-x を利用することを前提とする．

3.2 各経路における機密情報の拡散追跡

3.2.1 実現状況

2.1 節で述べたように，機密情報は，ファイル操作，プロセス間通信，および子プロセス生成により，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．現在，ファイル操作と子プロセス生成を行うシステムコールによる機密情報の拡散を追跡する処理は実現済みである．プロセス間通信については，今後設計と実装を進める．以降では，実現済みの追跡処理について述べる．

3.2.2 ファイル操作

ファイル操作に関連するシステムコールとして，open, read, write, および close をフックする．また，ファイル操作による機密情報の拡散を追跡するために，以下の 2 つの情報を取得する．

(1) カレントプロセスを特定する情報
(2) システムコールが扱うファイルを特定する情報

各システムコールをフックした際，システムコールを発行したプロセスが管理対象か否かを判定するために，カレントプロセスを特定する情報が必要である．カレントプロセスの特定には，プロセス ID（以降，PID と呼ぶ）を利用する．カレントプロセスの PID は，システムコールの入口でシステムコールをフックした際に取得する．

また，読み込むファイルが管理対象か否か判定する際や書き出し先のファイルを管理対象に加える際に，システムコールが扱うファイルを特定する必要がある．システムコールが扱うファイルの特定には inode 番号を利用する．inode 番号は，プロセス制御ブロックからファイル構造体まで辿っていくことに

より，取得する．このとき，ファイルディスクリプタ（以降，fd と呼ぶ）を利用することにより，inode 番号を特定する．fd は，open の場合，sysexit をフックした際に，戻り値から取得する．また，read，write，および close の場合，sysenter をフックした際に，引数から取得する．

3.2.3 子プロセス生成

子プロセス生成に関連するシステムコールとして，clone をフックする．また，子プロセス生成による機密情報の拡散を追跡するために，以下の 3 つの情報を取得する必要がある．

(1) 生成対象がプロセスかスレッドか判定する情報
(2) 親プロセスを特定する情報
(3) 子プロセスを特定する情報

clone は，新しくプロセスを生成するだけでなく，clone を発行したプロセス内でスレッドを生成する場合がある．同プロセス内のスレッドは，メモリやプロセスがオープンしているファイル情報といったリソースを共有する．このため，スレッド生成では，スレッドを生成したプロセスの外部に情報が拡散しない．一方，プロセス生成では，生成した側のプロセスが保持するリソースが生成された側のプロセスへ拡散する．

そこで，clone による機密情報の拡散を追跡する際，clone の生成対象がプロセスかスレッドを判定する必要がある．生成対象がプロセスかスレッドかを判定する情報には，clone のフラグを利用する．clone は，呼び出す際にフラグに `CLONE_THREAD` を指定することにより，スレッドを生成する．このため，clone のフラグを取得し，`CLONE_THREAD` がセットされているか否かにより，生成対象がプロセスかスレッドかを判定できる．フラグは，システムコールの入口で clone をフックした際に，引数から取得する．

また，親プロセスが管理対象の場合，子プロセスに機密情報が伝搬する可能性がある．そこで，親プロセスが管理対象か否かを判定するために，親プロセスを特定する情報を取得する必要がある．親プロセスを特定する情報には，親プロセスの PID を利用する．親プロセスの PID は，システムコールの入口で clone をフックした際に，取得する．

さらに，生成された子プロセスに機密情報が伝搬した可能性があると判定した際，子プロセスを管理対象プロセスに加える．そこで，子プロセスを管理対象プロセスに加える際に，子プロセスを特定する情報が必要である．子プロセスを特定する情報には，子プロセスの PID を利用する．clone は戻り値として子プロセスのスレッド ID（以降，TID）を返却する．新しくプロセスを生成する場合，PID と TID は同値である．このため，子プロセスの PID は，システムコールの出口で clone をフックした際に，戻り値から取得する．

表 1 評価環境

CPU		Intel Core i5-3470, 3.2GHz
OS	ゲスト	Debian 7.1.0 (Linux 3.2.46，32bit)
	ホスト	Fedora 18 (Linux 3.6.10，64bit)
メモリ	ゲスト	1,024MB
	ホスト	4,096MB
VMM		KVM-kmod-3.6

4 評価

4.1 評価項目

評価では，大別して，以下の 2 項目を測定した．

(1) 改変コード量
(2) オーバヘッド量

まず，機密情報の拡散追跡機能の実装工数を評価するため，文献 [2] の既存機能と VMM における拡散追跡機能のコード改変量を測定し，比較した．

また，機密情報の拡散追跡機能を KVM 内に実装することにより，仮想化によるオーバヘッドをはじめとした性能への影響が予測される．そこで，システムコール，マイクロベンチマーク，および応用プログラム（以降，AP と呼ぶ）の実行時間を測定し，文献 [2] と比較した．文献 [2] と VMM における拡散追跡機能の評価環境を**表 1** に示す．

4.2 改変コード量

4.2.1 評価方法

本評価では，評価基準として，論理 LOC（Lines Of Code）と修正箇所を含むファイルの総数を利用した．論理 LOC は，ソースコードの総行数から，記号だけの行，空白行，およびコメントのみの行を省いた総数である．論理 LOC の算出には，LocMetrics[6] を利用した．本測定により算出した値は，ファイル操作と子プロセス生成による機密情報の拡散追跡機能に関連するコードのみの値である．

また，文献 [2] は，Linux カーネル，VMM における拡散追跡機能は KVM を改変することにより機密情報の拡散追跡機能を実現しており，改変対象におけるソースコード規模が大きく異なる．そこで，それぞれにおいて，改変したファイルや新規に追加したファイルを有するディレクトリを測定対象とした．

表 2 論理 LOC と修正箇所を含むファイルの総数の比較

測定対象	行数			ファイル数		
	全体	追加行数	割合 (%)	全体	追加・修正ファイル数	割合 (%)
文献 [2]	47,222	763	1.61	101	14	13.9
VMM	35,559	898	2.53	49	10	20.4

具体的には，Linux カーネルの kernel/，fs/，および init/ディレクトリの直下，KVM の x86/ディレクトリの直下を測定対象とした．

4.2.2 評価結果

それぞれの論理 LOC と修正箇所を含むファイルの総数を測定した結果を**表 2** に示す．VMM における機密情報の拡散追跡機能は，文献 [2] に比べ，追加行数が 135 行増加している．これは，ファイル情報やプロセス情報といった OS の情報を VMM から取得する関数によるものである．改変対象のソースコード全体に対する追加行数が占める割合の差は，0.92%であり，極めて小さい差であるといえる．

また，4.2.1 項で述べたとおり，文献 [2] では改変したファイルや新規に追加したファイルは，複数のディレクトリに存在している．一方，VMM における拡散追跡機能では，単一のディレクトリにのみ存在している．文献 [2] は，機密情報の拡散に関係するシステムコール処理が記述された箇所を改変している．このため，改変箇所が複数ディレクトリにまたがっている．一方，VMM における拡散追跡機能は，VMM から全てのシステムコールを一元的にフックし，追跡処理を行う．このため，単一のディレクトリ以下を改変することにより，追跡処理を実現している．修正個所を含むファイルの総数も，VMM における拡散追跡機能は 10 個と，文献 [2] の 14 個に比べて約 70%に収まっている．

以上より，VMM における拡散追跡機能は，文献 [2] に比べ，僅かな追加の改変量で追跡処理を実現し，かつ改変する範囲を局所化しているといえる．

4.3 オーバヘッド量

4.3.1 評価方法

評価項目は以下の通りである．測定は，表 1 に示した仮想化環境のゲスト OS である Debian 上，および同性能のマシン上において直接動作させた Debian 上で行った．

(1) システムコール実行時間

文献 [2] においてオーバヘッドが測定されており，かつファイル操作や子プロセス生成に関するシステムコールである write，read，および close を測定対象とした．また，文献 [2] では子プロセスを生成するシステムコールとして，fork に生じるオーバヘッドが測定されているが，VMM における拡散追跡機能の評価環境では子プロセス生成には，clone が利用される．このため，clone のオーバヘッドを測定し，文献 [2] における fork に生じるオーバヘッドと比較した．

さらに，VMM における拡散追跡機能では，すべてのシステムコールをフックし，フックしたシステムコールが機密情報の拡散に関係するか否かを判定したのちに追跡処理を行う．このため，機密情報の拡散に関係するシステムコールだけでなく，機密情報の拡散に関係しないシステムコールにもオーバヘッドが生じる．そこで，機密情報の拡散に関係しないシステムコールとして getpid に生じるオーバヘッドも測定した．

(2) マイクロベンチマーク

本評価では，マイクロベンチマークとして，LMbench[7] を利用した．LMbench は，マシンの基本性能を測定するベンチマークツールである．VMM における拡散追跡機能がゲスト OS の性能に及ぼす影響を評価するために，基本的な OS 機能のレイテンシを測定した．

(3) AP 実行時間

VMM における拡散追跡機能では，ゲスト OS 上でシステムコールが発行されるたびに VMM へ処理が遷移するため，性能低下が予測される．そこで，AP 実行時間にどの程度の影響が出るかを評価するために，read と write を多数発行する処理として，Linux カーネルの bzImage のビルドに要する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

仮想化環境と同等の実環境における測定結果を**表 3-1**，仮想化環境における測定結果を**表 3-2** に示す．表 3-2 中の「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，

表 3 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）

3-1 実環境での測定結果

	機能導入前
write (file)	1.41
read (file)	0.57
close(file)	0.17
fork	18.17
getpid	0.0075

3-2 仮想化環境での測定結果

	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	(管理対象資源の操作)
write (file)	4.51	6.23	6.82	2.31
read (file)	3.93	5.26	5.43	1.50
close(file)	0.151	1.24	1.29	1.14
clone	168.88	200.36	198.20	29.32
getpid	0.0069	0.0075	0.0084	0.0015

VMM における拡散追跡機能を導入済みの環境での測定値であり，それぞれ，管理対象ファ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の測定値である.「オーバヘッド」の項目は,（管理対象資源の操作における測定値－仮想化環境の機能導入前における測定値）で算出した値である．仮想化環境での「オーバヘッドの割合」は，上記のオーバヘッドが管理対象ファイルを操作する場合の測定値に占める割合である．また，文献 [2] の測定と本測定のオーバヘッドの割合を **図 2** に示す.

機密情報の拡散に関係するシステムコールである write，read，close，および clone は，オーバヘッドの割合がそれぞれ 51.1%，38.1%，758.1%，および 17.4%と比較的大きめの値となっている．実時間も，1.1 ～ 29.3μs と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 70000 倍以上の値となっている．フックしたシステムコールが機密情報の拡散に関係すると判定した場合，引数を保存し，一旦処理をゲスト OS に返却した後，再度 sysexit をフックし，戻り値を取得する．その後，引数や戻り値をもとに，inode 番号などの OS 情報を取得し，取得した情報をもとに機密情報の追跡処理を行う.

一方，機密情報の拡散に関係しないシステムコールである getpid は，オーバヘッドの割合が 22.7%とやや大きめの数値であるが，実時間は，0.0015μs と比較的小さい値である．VMM における拡散追跡機能は，フックしたシステムコールが機密情報の拡散に関係しないと判定した場合，何もせずにゲスト OS へ処理を返却するため，機密情報の拡散に関係しないシステムコールは，比較的小さいオーバヘッドになっている.

上記のように，機密情報の拡散に関係するシステムコールは，機密情報の拡散に関係しないシステムコールに比べ，多数の処理を行うため，オーバヘッドが大きくなっている.

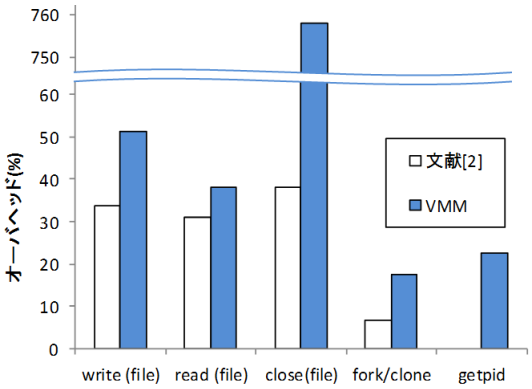

図 2 システムコールに生じるオーバヘッド割合の比較

表 4 LMbench による測定結果

	実環境 (μs)	仮想化環境	
		機能導入前 (μs)	機能導入後 (μs)
null call	0.05	0.05	1.07
null I/O	0.09	0.09	1.12
stat	0.36	0.37	1.49
open clos	0.6	0.63	2.88
slct TCP	2.08	2.1	3.15
sig inst	0.17	0.17	1.19
si hndl	0.96	0.97	1.14
fork proc	62.1	765	882
exec proc	195	1915	2246
sh proc	473	4336	5069

4.3.3 マイクロベンチマークの評価結果

LMbench により，OS 機能のレイテンシを測定した結果を**表 4** に示す．また，実環境での測定結果を 1 とした際の各測定環境での測定結果の比率を**図 3** に示す．まず，fork proc, exec proc, および sh proc は，実環境に比べ，測定結果に 820~4596μs の影響が出ていることが分かる．ただし，その多くが仮想化によるものであり，上記の測定項目において，VMM における拡散追跡機能の影響は，仮想化の影響に比

表 5　bzImage のビルド時間

			実行時間 (s)	ユーザ時間 (s)	システム時間 (s)
実環境	機能導入前		429.047	376.939	22.859
仮想化環境	機能導入前		454.321	399.230	22.834
	機能導入後	管理対象ファイル数 0	624.434	447.652	124.573
		管理対象ファイル数 10	651.262	456.339	141.909
	オーバヘッド (管理対象ファイル数 10)		196.971	57.109	119.075
	オーバヘッドの割合 (%)		30.2	12.5	83.9
文献 [2]	オーバヘッドの割合 (%)		0.46	-0.02	3.7

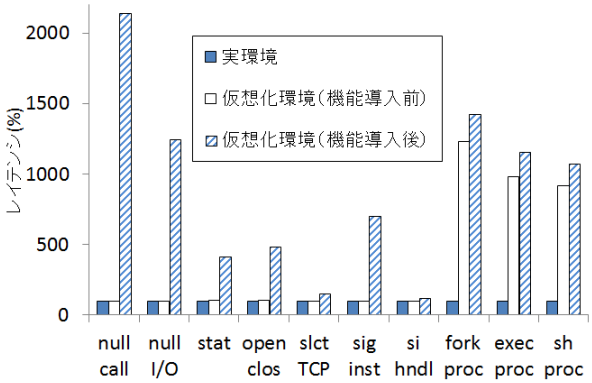

図 3　LMbench による測定結果の比率

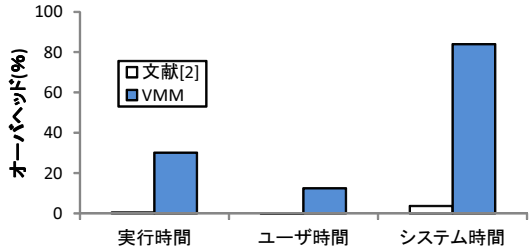

図 4　bzImage ビルド処理時間に生じるオーバヘッドの割合

べると小さいといえる．

また，null call, null I/O, stat, open clos, および sig inst は，実環境での測定結果からの比率が 414~2140%と大きい値となっている．これは，元々の測定結果が小さいことにより，相対的に比率が大きくなっているためである．実測値は，約 1~2μs 程度の影響におさまっている．ただし，open clos は，2.28μs と，上記の測定項目の中では，最も影響が大きい．これは，open と close が機密情報の拡散に関係するシステムコールとして追跡処理の対象となっているためである．

以上から，VMM における拡散追跡機能が OS 機能の性能に与える影響は，仮想化による影響に比べると小さい，あるいは，実測値でみると小さいといえる．ただし，機密情報の拡散に関係するシステムコールを利用する処理は，機密情報の拡散に関係しないシステムコールを利用しない処理に比べ，性能への影響が大きい．

4.3.4　AP 実行時間の評価結果

測定結果を**表 5** に示す．表中の「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「管理対象ファイル 0」と「管理対象ファイル 10」の各項目は，VMM における拡散追跡機能を導入済みの環境での測定値であり，それぞれ，管理対象ファイルを登録しない場合と 10 個のファイルを登録した場合の測定値である．「オーバヘッド」の項目は，(管理対象ファイルが 10 個の場合における測定値－仮想化環境の機能導入前における測定値) で算出した値である．仮想化環境での「オーバヘッドの割合」は，管理対象ファイルが 10 個の場合における測定値に占める上記のオーバヘッドの割合である．また，文献 [2] と本評価で測定されたオーバヘッドの割合を**図 4** に示す．

表 5 より，実行時間におけるオーバヘッドの割合が既存機能の約 65 倍となっている．これは，bzImage のビルド中にオーバヘッドの大きい read や write が多数発行されることにより，処理全体のオーバヘッドも大きくなるためと考えられる．また，ユーザ時間に比べ，システム時間のオーバヘッドの割合が大きいことが分かる．システム時間は，カーネル内の処理時間を意味する．VMM における拡散追跡機能は，read や write など，機密情報の拡散に関係するシステムコールをフックする際，カーネル内の処理である sysexit 中にゲスト OS の処理をフックし，inode 番号などの OS 情報を取得する．この OS 情報の取得がシステムコールのオーバヘッドの主な要因であるため，ユーザ時間に比べ，システム時間のオーバヘッドが大きくなっていると考えられる．このため，機密情報の拡散に関係するシステムコールの発行回数に比例して，処理時間は遅くなると推測される．

以上から，VMM における拡散追跡機能は，AP

実行時間に大きな影響を及ぼすといえる．このことから，実用に向けての課題として，オーバヘッドの削減が挙げられる．

5 関連研究

TightLip[8] 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，Doppelgangers と呼ばれるプロセスに置き換える．このとき，Doppelgangers は，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．

また，モバイル端末において，情報漏えいを防止する手法が研究されている．文献 [9] は，機密情報をセキュリティマネージャと呼ばれる機構により管理する．セキュリティマネージャは，機密情報が端末外に出力される場合，ユーザに確認を求め，ユーザの承諾があったときのみ，実際に出力することにより，機密情報の漏えいを防止する．TaintDroid[10] は，機密情報に Taint と呼ばれるタグを付与し，モバイル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．

6 おわりに

VMM からゲスト OS 上のファイル操作や子プロセス生成による機密情報の拡散を追跡する機能の実現方式を述べた．また，実現方式に基づいて実装し，実現した機能を評価した．

改変コード量の評価では，VMM における拡散追跡機能は，文献 [2] に比べ，僅かな追加の改変量で追跡処理を実現し，かつ改変する範囲を局所化できることを示した．

システムコールのオーバヘッド評価では，機密情報の拡散に関係しないシステムコールには 0.0015μs と僅かなオーバヘッドしか生じないことを示した．機密情報の拡散に関係するシステムコールには，1.1 ~29.3μs と，機密情報の拡散に関係しないシステムコールに比べ，大きなオーバヘッドが生じることを示した．

マイクロベンチマークを用いた評価では，仮想化の影響を大きく受ける機能では，本手法による性能への影響が相対的に小さく，そうでない機能については，オーバヘッドが約 1~2μs と小さいことが分かった．

AP 実行時間の評価では，VMM における機密情報の拡散追跡機能は，機密情報の拡散に関係するシステムコールを発行する処理に対して，実行時間に約 30%のオーバヘッドを生じることを示した．

今後の課題として，プロセス間通信をはじめとしたすべての機密情報の拡散を追跡すること，およびオーバヘッドの削減がある．

謝辞 本研究の一部は，平成 25 年度公益財団法人ウエスコ学術振興財団学術研究費助成による．

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に関する調査報告書 ~ 個人情報漏えい編 ~, http://www.jnsa.org/result/incident/2012.html.

[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] 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）.

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

[5] Chen, Peter M. and Noble, Brian D.: When Virtual Is Better Than Real，*Proc. HotOS*，pp.133–138 (2001).

[6] locmetrics.com: LocMetrics，http://www.locmetrics.com/.

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

[8] Yumerefendi, A. R., Mickle, B. and Cox, L. P.: TightLip: Keeping Applications from Spilling the Beans, *Proc. NSDI*, pp.12–12 (2007).

[9] 上松 晴信，可児 潤也，米山 裕太，川端 秀明，磯原 隆将，竹森 敬祐，西垣 正勝 : Android OS におけるマスカレーディングポインタを用いたプライバシー保護 (その 2)，情報処理学会研究報告，Vol.2013-DPS-154，No.57，pp.1–8 (2013).

[10] 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, *Proc. OSDI*, pp.1–6 (2010).