情報処理学会論文誌 Vol.54 No.2 1–11 (Feb. 2013)

仮想化技術による安全なファイルアクセスログ外部保存機構

安藤 類央[1] 橋本 正樹[2] 山内 利宏[3]

受付日 2012年5月14日, 採録日 2012年11月5日

概要：近年，計算機が攻撃を受けた後で，その被害を抑制する技術や被害を確実に把握する技術の重要性が高まっており，仮想化技術によりゲスト OS の挙動を観測することで，汎用 OS の安全性を高める研究が盛んに行われている．しかしながら，ゲスト OS の観測については，計算機に被害を与えるマルウェアの解析の高粒度化のためにメモリ検査が行われる場合が殆どであり，重要な観測項目であるファイルアクセスを捕捉するための手法は確立されていない．本論文では，ファイルシステムのフィルタドライバをハイパーバイザーと観測可能にすることで，ゲスト OS のファイルアクセスを捕捉し，アクセスログを外部のハイパーバイザーに安全かつ確実に保存することが可能である．また，ファイルアクセス観測と併せることで，従来のメモリアクセスの監視と解析では検出が困難なマルウェアに関しても検出を容易にし，同時に従来のメモリ観測解析による検出範囲の拡大を可能にしている．

External Storage Mechanism for Preserving File Access Log with Virtualization Technology

Ruo Ando[1] Masaki Hashimoto[2] Toshihiro Yamauchi[3]

Received: May 14, 2012, Accepted: November 5, 2012

Abstract: Recently, it is more important to grasp and control the damage of attacks safely, so much research has been done to increase the security of the general-purpose OS by observing the behavior using virtualization technology. In this paper, we propose a mechanism to observe and logging the file access in guest OS from virtual machine monitor using the inter-domain communication by the filter driver. Our mechanism can be applied independently of the implementation of virtual machine monitor. By hooking file accesses in the guest OS, log messages are transferred and stored to the virtual machine monitor, so our approach is effective from the viewpoint of preservation of the log. We show the design and implementation of our mechanism for both Xen and KVM. Furthermore, we report the results of measuring the performance when accessing files as evaluation.

1. はじめに

近年，多種のマルウェアが開発されており，中には未知の脆弱性を利用したゼロデイ攻撃を行うものもあるため，既存のアンチウイルスソフトウェアでは，確実にすべてのマルウェアを検知することが難しい．また，標的型攻撃等により，クライアント計算機が被害を受けることも増えており，マルウェアから計算機を保護する技術が重要となっている．さらに，計算機が攻撃を受けるのを防止するだけ

[1] 情報通信研究機構
National Institute of Information and Communications Technology
[2] 情報セキュリティ大学院大学情報セキュリティ研究科
Graduate School of Information Security, Institute of Information Security
[3] 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

でなく，攻撃を受けた後でその被害を抑制する技術や，被害を確実に把握するための技術の重要性も高まっている．

このような背景の中，仮想化技術を利用することにより，ゲスト OS から隔離された外部環境から，より安全にゲスト OS の挙動を観測し，汎用 OS の安全性を高める研究が盛んに行われている．仮想マシンモニタによる仮想マシンの観測は，VMI (Virutal Machine Introspection)[1] と呼ばれており，観測系の隔離と，そこからの検査・介入により，ゲスト OS への攻撃を検知・防止することを可能とする．VMI は，仮想マシン内に実装した観測系によりイベントを捕捉，通知する内部観測手法と，仮想マシンモニタ外部でのみ取得できる情報から観測を行う外部観測手法の２通りに分類できる．

この内，外部観測には，ゲスト OS の修正が不必要であること，ゲスト OS から観測系が不可視なこと等の利点があるが，ゲスト OS 内部の情報がホスト OS 側に不足することに起因する課題が存在する．特に，ファイルアクセスの観測という観点からすると，外部観測では，I/O レイヤの上に位置するファイルアクセスログの捕捉と，仮想マシンモニタ側での再構成が困難となる．もちろん，ゲスト OS のメモリアクセスの観測と捕捉という観点から低レベルな観測を達成することが可能であるが，ファイルアクセス API 等の高レベルな観測からマルウェアの感染前の予兆などを検出することは難しい．

一方で，内部観測では，修正の内容次第では，ゲスト OS 内部の情報をホスト OS が柔軟に取得可能で，これにより，例えば，ホスト OS のアンチウィルス機能や汎用の侵入検知・防止システムが持つ豊富な機能と十分に連携するようなことが可能となる．その反面で，観測対象とするゲスト OS や仮想マシンモニタの修正が必要となるため，仮想化環境の構成毎に実装手法の検討が必要となることが課題で，さらには，修正に伴う追加処理の影響による性能劣化を可能な限り低減する必要がある．

現況の関連研究では，マルウェアの解析の高粒度化のためにゲスト OS のメモリの観測を行うものが殆どであり，ゲスト OS のファイルアクセスを捕捉するための試みは殆ど行われていないのが現状である．しかしながら，マルウェアの中にはトロイの木馬やアドウェア，スパイウェアなど，メモリアクセス解析の高粒度化だけでは対応できないものが多く，その際はファイルアクセスと併せて観測しないと検出が難しいものが多い．それにも関わらず，現況では，仮想マシンのファイルアクセスをハイパーバイザー側で捕捉し，時系列データから検出を行う研究は存在しない．

そこで本論文では，Windows OS にフィルタドライバを実装し，ゲスト OS 内でファイルアクセスが発生した際に仮想マシンモニタとドメイン間通信を行うことで，ホスト OS がゲスト OS のファイルアクセスを内部観測によって達成する手法を提案する．提案システムは，外部観測が孕む，仮想マシンのメモリ観測と解析による状態観測のコストを回避した上で，仮想マシンのサスペンドを行わずに，観測を可能にする．また，提案システムでは，ファイルアクセスに関して，コールバックステータスも含めたすべての状態を捕捉することが可能であり，ファンクションドライバとアプリケーションの間で動作するため軽量な稼動が可能になる．

提案システムでは，ファイルシステムのデバイスドライバを用いることで，ユーザ空間より安全なカーネル空間でのアクセスログの取得を行うことが可能である．これにより，ユーザ空間上で可能な観測系への攻撃はすべて防ぐことができると同時に，カーネル空間へ侵入してくる攻撃も検出することが可能になる．また従来の syslog への転送などと比較して，ログの転送路をユーザ空間に経由しないことでより安全なログの取得を行うことができる．さらに，デバイスドライバを用いることで，ユーザプロセスでの観測に比べ，書き込み，読み込みの前後，ファイル属性の変更などより高粒度なアクセスログを取得することができる．これにより，従来の研究では提案されてこなかった仮想マシンのファイルアクセスの時系列データを外部のハイパーバイザーに安全，確実に保存するシステムを構築し，またこれによりファイルアクセスの捕捉から，マルウェアを検出範囲の拡大が可能になった．

以下に，本論文の構成を示す．はじめに，第 2 章では，仮想マシンモニタによる仮想マシンの観測について概観し，関連研究を外部観測によるものと内部観測によるものにわけて説明した後に，本研究の目的を説明する．次に，第 3 章では，提案手法の設計として，基本方式とその中核となるドメイン間プロトコルについて説明する．続いて，第 4 章では，提案手法の具体的な実現方式として，ゲスト OS としての Windows OS と，仮想マシンモニタとしての Xen，KVM に対する実装を説明し，第 5 章では，提案手法に対する評価実験とその考察について述べる．最後に，第 6 章で，本論文をまとめる．

2. 研究の背景

本章では，はじめに，仮想マシンモニタの利点と VMI の起源について，具体例を示しながら説明する．その後，VMI を外部観測手法によるものと内部観測手法によるものに大別した上で，各々の実装方法を説明する．また，関連研究として，具体的な研究例を取り上げながら各々の課題を説明する．最後に本研究の目的を説明する．

2.1 仮想マシンモニタによる仮想マシンの観測

仮想マシンモニタは，仮想マシンとハードウェアの中間に位置する小さなソフトウェアで，リソース管理やスケジューリングを仮想マシンに対して行う．その代表的な実

装例としては，Xen[2]，KVM[3]，VMware[4]，Windows 7 (XP モード利用時)[5] 等がある．仮想マシンモニタにより享受できる利点は，第一に，運用面で，複数の仮想マシンを一元管理できること，即ち，稼動状況にあわせて割り当てるリソースを動的に伸縮できることがある．第二は，仮想マシンのセキュリティ強化で，仮想マシンモニタを用いることで，複雑化し脆弱性を内包するようになった昨今の汎用 OS 上でアクセス制御を強化したり，マルウェアから検出不可能な観測・解析器を構築できることである．

Xen の Split Kernel Driver は，その代表例で，仮想マシンとハイパーバイザの間で共有メモリを設置し，イベントチャンネルを用いて入出力を仮想化する．また，この応用として，XenAccess[6] では，Split Kernel Driver である blktap ドライバを用いて仮想マシンのイベントの修正し，システム全体の負荷を軽減することができる．また，sHype[7] では，仮想マシンのスナップショット生成・破棄時や仮想割り込み発生時に，RBAC[8], [9] や TE[10], MLS[11], [12], [13] といったセキュリティポリシにしたがった動作を強制することができる．

しかしながら，これらの手法は総じてブロックデバイスへのアクセスを捕捉するものであり，ゲスト OS のファイルアクセス自体を観測・検証するものではない．また，これらの手法はゲスト OS を外部から観測するものであり，Windows をゲスト OS として稼動させる場合，Windows に内在するファイルシステムのセマンティクスの複雑さから，外部観測により詳細なイベントの観測を行うことは難しい．提案システムでは，ハイパーバイザーとの通信用に独自の修正を加えた Windows のファイルシステムのドライバを用いることで，ゲスト OS のファイルファイルアクセスを高粒度かつ低負荷に観測するための手法を実装している．また同時に，提案システムはアクセス制御の強化や観測の高粒度化に加えて，観測範囲の向上を企図している点に新規性がある．

仮想マシンモニタを用いて仮想マシンを監視し，観測，防御，隔離などの処理を行う VMI の概念は，Garfinkel の研究 [1] により提示された．この中で，VMI は，仮想マシンから仮想マシンモニタ側のコードを修正できないこと，仮想マシンモニタ側から仮想マシンのすべての状態を観測できること，仮想マシンから発行されるコードを捕捉できることの 3 点により，不正アクセス検知と防御に有効であると指摘されており，本研究を参照した仮想マシンの観測についての論文が多数発表されている．

2.2 関連研究

VMI による観測方法には，外部観測による方法と内部観測による方法がある．外部観測として代表的な手法は，仮想マシンの状態を外部から一定時間毎に取得し，情報を抽出するもので，メモリのスナップショットを取得解析する Volatality[14] や，Revirt[15] がある．外部観測としては，仮想マシン内部でリソースアクセス等のイベントが起きた場合に関連する情報を引き出す方法があり，代表的な研究例として，Lares[16]，Xenprobes[17]，VMScope[18] がある．

外部観測と内部観測の具体的な実装方法としては，内部観測 (in-the-box) 方式と外部観測 (out-of-the-box) 方式の 2 通りあり，これらは観測機構の攻撃者側からの検出可能性とセマンティックギャップ [19] の解消可能性[*2]，実装に係るコストのトレードオフ [20] との関連性から論じられることが多く，現状では，観測対象の仮想マシン内部から観測機構が検出困難であるという理由から，外部観測方式が採用されることが多い．一方で，外部観測方式には，内部観測方式と比べて，観測可能な情報量が低下する課題がある．

外部観測を行うものとして，Volatility は，仮想マシン上のメモリを解析し，情報を検索するためのツールである．Xen の Split Kernel Driver を利用することで，Xen におけるドメイン間の通信を実現するが，仮想マシン上で生成されるファイルアクセスの観測をサポートしていない．また，Revirt は，仮想マシンモニタの拡張により仮想マシンに対する不正侵入を分析できるが，そのためには，仮想マシンの一時停止とスナップショット取得が必要となるため，Volatility 同様に，時系列でのイベントの観測の粒度は高くない．

内部観測を行うものとして，Patagonix[21] と Lares は，仮想マシン上のイベントを検知するシステムで，仮想マシンモニタと仮想マシン間のセマンティックギャップ解消を実現している．Patagonix は，仮想マシン上でのバイナリ実行を検知し，どのようなタイプのバイナリがロードされ，実行されたのかを特定する．Lares では，アクティブモニタと呼ばれるセキュリティ・モジュールを仮想マシンに展開し，メモリ操作を監視することで，安全な仮想マシン間の隔離を実現する．アクティブモニタによる仮想マシンの観測手法は，提案システムと似たものであるが，Lares は，メモリの不正利用検知と安全な隔離を目的に設計されており，リアルタイムのファイルアクセスを観測するように設計されていないが，ワームやトロイの木馬等の悪性のソフトウェア検知を目的とした場合は，後者が特に重要となる．

また，別の関連研究として，Quynh[22] と King ら [23] では，仮想マシンモニタによってログ取得とデバックを行うアーキテクチャが提案されている．即ち，文献２２では，仮想マシンモニタ技術が，サーバ集約と同時にログ取得アーキテクチャを集約できることを示し，また，King ら

[*2] セマンティックギャップとは，仮想マシン内部のリソースアクセスに関連するセマンティック情報を，ハイパーバイザ側で得られる情報から再構築できないことを言う．即ち，セマンティックギャップの解消とは，ハイパーバイザが仮想マシン上で起こるリソースアクセスのセマンティックを再構築可能することである．

は，仮想マシン上のプログラムを任意のタイミングで再実行するために，仮想マシンのスナップショット取得と再実行を行うユーティリティを提案した．しかし，これらのシステムは，メモリ解析によるデバッグのみを対象に設計されている．

これらのメモリのイントロスペクションの手法は総じて，ゲスト OS の中でも比較的低レイヤで実行されるマルウェアの解析自体の高粒度化を目指しているものであり，ファイルアクセス観測と合わせなければ検出が困難なマルウェアに重点を置いていない．結果としてマルウェアの中でも重大な被害を及ぼすトロイの木馬やアドウェア，スパイウェアなどの大部分のマルウェアの検出に関しては困難であるか，非効率な場合が多い．しかしながら，仮想マシン，特に Windows OS のファイルアクセスの捕捉を行うことで時系列データを生成し，マルウェアの検出を行う研究は現時点では存在しない．このような現況に対し，本論文の提案システムではゲスト OS のファイルアクセスの捕捉に重点を置き，メモリ観測とあわせることで，マルウェアの検出精度の向上と共に，検出範囲の拡大を可能にしている．また，ハイパーバイザーとの通信機能を持つファイルシステムのデバイスドライバを実装したことで，ファイルアクセスログをゲスト OS の外部に安全かつ確実に保存・検証することができる．

2.3 研究の目的

本研究の目的は，Windows OS にフィルタドライバを実装することで，仮想化環境において，ゲスト OS としての Windows OS のファイルアクセスをホスト OS が内部観測により捕捉する手法を提案し，これによって，軽量かつ VM のサスペンドを伴わないファイルアクセス観測とその安全な外部保存を実現することである．

従来の観測手法は，ゲスト OS の修正が不要である点が重視されることで，第 2.2 節で例にあげた Patagonix や Lares 等のような仮想マシン側を修正しない外部観測によるものが主流であったが，外部観測には，ゲスト OS 内部の情報がホスト OS 側に不足する問題がある．この原因は，第一に，ファイルアクセスに関する API 呼び出しをホスト OS が検知・解析できないことで，第二に，仮想マシンのメモリ挙動観測ではファイルアクセスを捕捉できないことにあり，この問題は，ファイルアクセスログの安全な取得や，それによるマルウェア感染前の予兆検出等を目的とした場合には，より大きな問題となる．

従って．この目的のためには外部観測の方が適していると言えるが，Volatility や Revirt 等の外部観測による既存研究では，各々想定している利用用途が異なるため，いずれも時系列のアクセスデータの解析によるのマルウェアの検出には粒度が不十分であるし，仮想化環境の構成に強く依存した実装手法により実現されているため，適用範囲が非常に限定される．本研究では，特にこれらの課題に着目し，内部観測を用いることで VM をサスペンドさせることなく，アクセスのログの取得を実現可能とする手法を検討する．また同時に，提案手法が十分実用に耐える程度に軽量で，ゲスト OS と仮想マシンモニタに対する修正が少量で済むことを，実際に提案手法を既存 OS と仮想マシンモニタに実装した上で，定量的に評価し，確認する．

即ち，本論文では以下の点を明らかにする．

(i) ファイルアクセス観測のためのドメイン間通信プロトコルの設計
(ii) 既存 OS と仮想マシンモニタに対する (i) の具体的な実装方式
(iii) (ii) による負荷が現実的に許容可能であること

提案手法によって期待される効果としては，フィルタドライバを用いた低負荷の実時間観測により，仮想マシンをサスペンドすることなく攻撃の予兆や早期発見が可能となることが期待できるし，レジスタを用いたログ文字列の転送により，仮想マシンモニタの実装方法に依存せずに適用できる．また，ログ文字列はゲスト OS 側の仮想化されたメモリやハードディスクに保存されることなく，仮想マシンモニタ側に転送され格納されるため，ログの保全という観点からも効果的である．さらには，提案するドメイン間通信プロトコルは，汎用レジスタとデバッグレジスタを提供する仮想マシンモニタであれば適用可能であり，フィルタドライバは，各種 Windows OS (XP,Vista,7) に実装可能である．

3. ファイルアクセス観測機構の設計

本章では，提案手法を実現するための基本方式について，概要と特徴を説明する．また，提案手法の中核となるドメイン間通信について，概要と具体的なプロトコルを説明する．

3.1 基本方式

提案手法は，ゲスト OS にドライバを組み込むことで，ファイルアクセスログを取得する内部観測によるファイルアクセス検知手法である．ドライバは，後述するドメイン間通信を用いることで，仮想マシンモニタに取得したログを送信するもので，提案手法では，ゲスト OS のフィルタドライバによって転送するログを汎用レジスタに格納した上で VM_EXIT を発生させ，仮想マシンモニタの対応するハンドラを起動させる．その後，ハンドラにおいては，汎用レジスタから文字を取得し，仮想マシンモニタ上でログを保存する．仮想マシンモニタは最上位の特権で動作するため，ゲスト OS のリソースアクセスを捕捉可能であり，安全な観測を効果的に適用できる．

提案手法の特徴として，異なる実装方式の仮想マシンモニタに対して，同じドメイン間通信方式を適用できること

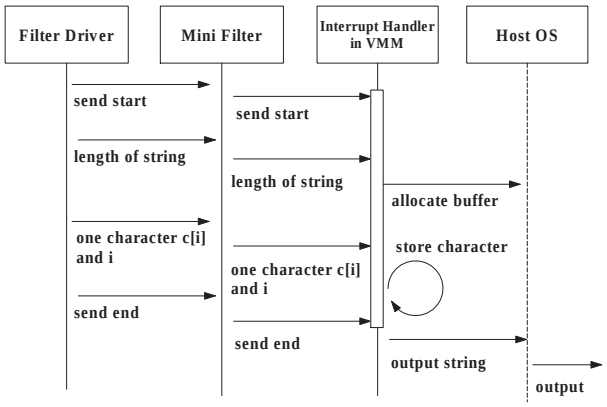


図 1 ドメイン間通信プロトコル
Fig. 1 Inter-domain communication protocol

がある．提案手法は，IA-32 アーキテクチャのデバッグレジスタ，及びゲスト OS のドライバを利用して，ドメイン間通信を実現することで，ログの取得を実現している．このため，デバッグレジスタを持つアーキテクチャで，ファイルアクセスをフックできるドライバの組み込みが可能な OS であれば，適用可能である．本論文では，ゲスト OS として Windows OS を，仮想マシンモニタとして Xen と KVM を対象として，ドライバとドメイン間通信を設計し，実装した．

3.2 ドメイン間通信プロトコル

図 1 に，ドメイン間通信プロトコルの概要を示す．提案手法のドメイン間通信プロトコルでは，エンティティとして，フィルタドライバ，ミニフィルタドライバ，仮想マシンモニタ内部の割り込みハンドラ，及びホスト OS の出力部分の 4 点がある．

フィルタドライバとミニフィルタドライバは，ゲスト OS として動作する Windows 内部に存在する．ここで，フィルタドライバ，ミニフィルタドライバの定義と関係を述べる．

Microsoft 社の定義では，ドライバには，フィルタマネージャ，ミニフィルタ，ファンクションドライバの 3 種類がある．フィルタマネージャは，Microsoft 社が提供するファイルシステムフィルタドライバである．ミニフィルタは，フィルタマネージャモデルに応じて開発されたドライバで，フィルタマネージャよりさらに機能を簡素化したものである．また，フィルタドライバの下稼動するドライバをファンクションドライバという．ファンクションドライバは，所謂通常のドライバで，実際の I/O のディスパッチ等を行うものである．フィルタドライバはドライバ本来の働きはなく，ロギングやチューニングに使われるため，軽量な動作を達成することができる．

また，フィルタドライバは，別名 intermediate driver と呼ばれ，Windows カーネルとファンクションドライバと呼ばれるデバイスドライバの間で，Microsoft 社が提供しているフィルタマネージャと協調して動作する．フィルタドライバを用いることで，軽量な実装でファイルアクセスを捕捉することが可能になり，また，カーネル空間で動作する Zw 系列の API をフックするドライバと比較して，ファイルシステムのほぼすべての構造体を引数にし，ファイルアクセスに関するの十分な情報を得ることが可能である．図 1 に示すように，フィルタドライバは仮想マシンモニタへ 4 つのメッセージを送信する．送信するメッセージは，通信開始，送信する文字列の長さ，1 文字ずつに分割されたログ，及び通信終了のメッセージである．これらのメッセージを用いてホスト OS 側でログ文字列を再構築する．

ドメイン間の通信プロトコルを以下に述べる．

1. 仮想マシン側のフィルタドライバは，ログ文字列の 1 文字ずつに対し，手順 2 以降の処理を繰り返す．
2. 送信する文字をアスキーコードに変換し，汎用レジスタに格納する．
3. フィルタドライバは，デバッグレジスタを変更する．
4. デバッグレジスタの変更により VM_EXIT が発生する．
5. 仮想マシンモニタ側のデバッグレジスタハンドラが起動される．
6. デバッグレジスタハンドラの修正部分により，ゲスト OS が汎用レジスタの値を格納する．

手順 1 ではログ文字列をアスキーコードに分割する．例えば，”test string”は以下のようになる．

```
"test string" -> 74, 45, 73, 74, 20, 73,
 74, 72, 69, 6E, 67
```

これらのアスキーコードのうち，例えば 1 文字目の 74 などは EAX レジスタに以下のように格納される．

```
__asm{
mov eax, 74
}
```

提案手法では共有メモリを用いずに，レジスタを操作することでドメイン間通信を行うことが可能なので，仮想マシンモニタの実装に依存せずにゲスト OS 内で発生したログ文字列の転送を行うことできる．

また，仮想マシンモニタとゲスト VM の間で，共有メモリの確保と処理を行わずにレジスタを用いた通信を行うという点から，Linux についても本手法が適用できる．すなわち，Linux のシステムコールは容易にフック可能であり，上記のレジスタを操作するアセンブラ命令を Linux のソースコードの適切な箇所に追加することで，仮想マシンモニタ側の実装は一切変更せずに，Windows OS で実装した提案システムと同じ機能を実現することができる．

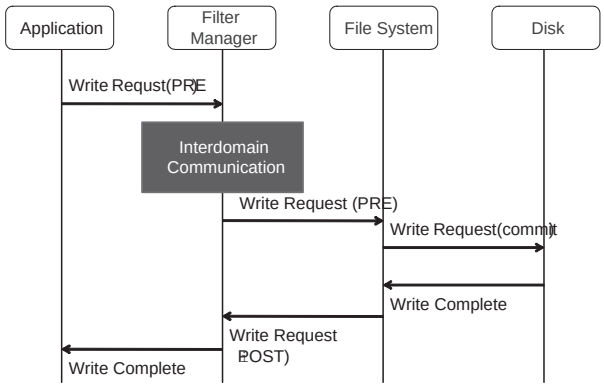


図 2 Windows OS のフィルタマネージャによるファイルアクセス
Fig. 2 File Access using Filter Manager in Windows OS

type		FLT_PREOP_CALLBACK_STATUS
function		Obtains file information structure Control file access Returns value for file access
arg	**1**	PFLT_CALLBACK_DATA Data
	2	PCFLT_RELATED_OBJECTS FltObjects
	3	PVOID *CompletionContext
return	**1**	FLT_PREOP_SUCCESS_WITH_CALLBACK (succeed)
	2	FLT_PREOP_SUCCESS_NO_CALLBACK (succeed)
	3	FLT_PREOP_COMPLETE (succeed/fail)
failure		Return: FLT_PREOP_COMPLETE Set Data->IoStatus.Status error

図 3 I/O 操作のコールバックステータスに関する構造体
Fig. 3 Structure for the Callback Status of I/O Operation

4. 既存 OS と仮想マシンモニタに対する提案手法の実現方式

本章では，提案手法の具体的な実現方式として，ゲスト OS としての Windows OS と，仮想マシンモニタとしての Xen，KVM に対する実装を説明する．

4.1 実装の概要

本論文では，提案手法が異なる方式の仮想マシンモニタに適用できることを示すため，Xen 3.4.0 と，Linux 2.6.37 上で稼動する KVM に提案手法を実装した．

Xen は，独自のブートローダやスケジューラを持つ仮想マシンモニタで，ドメイン 0 と呼ばれるホスト OS を稼動させる．ゲスト OS の稼動とドメイン間の通信には，イベントチャンネルやグラントテーブルなどの仕組みを用いる．一方，KVM は Linux 内部にドライバとして実装され，スケジューラやブートローダは Linux 内部のモジュールが担当する．このため，Xen と比較して実装はコンパクトなものになっている．VMI には Xen を用いたものが多いが，KVM を用いた外部観測の方法も提案されている [20]．ゲスト OS としては，Windows 7 SP1 を用いた．

4.2 Windows OS に対する修正

本節では，提案手法における Windows OS 側での修正について述べる．具体的には，フィルタドライバの構造と，ファイルアクセス時の関数と構造体について述べる．

Windows OS におけるファイルアクセスの様子を**図 2** に示す．フィルタマネージャは，アプリケーションとファイルシステムの間に位置し，Write Request (PRE) の情報を元にドメイン間通信を行う．フィルタドライバが処理する書き込み要求には，PRE と POST がある．これらは，実際に書き込みのコミットが行われる前後に発行される関数である．

図 3 に，I/O 操作のコールバックステータスなどの仕様を示す．フィルタドライバによる書き込みの要求は，CREATE, CLOSE, CLEANUP, READ, WRITE，及び SET_INFORMATION の 6 種類である．書き込みの要求を行う各関数は，以下に示す構造体を引数にとるため，構造体を参照することで発行したプロセス ID やフルパス名などの情報を取得することができる．

```
typedef struct _FILE_ACCESS_DATA{
    BOOLEAN IsDirectory;
    UNICODE_STRING  FullPath;
    ULONG AccessRecord;
    ULONG AccessAttributes;
    ULONG ProcessID;
    PWCHAR ProcessName;
    ULONG ProcessAttributes;
} FILE_ACCESS_DATA, *PFILE_ACCESS_DATA;
```

提案システムは，上記構造体を利用して仮想マシンのファイルアクセスを捕捉する．図 2 から，ファイルへの書き込み，読み込みのうち，PreRead，PreWrite を検知し，FullPath, ProcessID, ProcessName をログファイルに記録する．下記は，評価実験に出力されたログの抜粋である．

```
May  3 08:58:21 localhost kernel: [log][vm1]
;filerw;PreWrite;11;40;processId;1112;SVCHO-
ST.EXE;C:\WINDOWS\PREFETCH\NET1.EXE-029B9DB4
.PF
May  3 08:58:25 localhost kernel: [log][vm1]
;filerw;PreRead;11;40;processId;1988;WMICL-
ISV.EX;C:\WINDOWS\SYSTEM32\DRIVERS\ETC\HOSTS
.PF
```

この例により，May 3 08:58:21 に，SVCHOST.EXE(プロセス番号 1112) が C:\WINDOWS\PREFETCH\NET1.EXE-029B9DB4.PF に書き込み (PreWrite) を行っていることが確認できる．

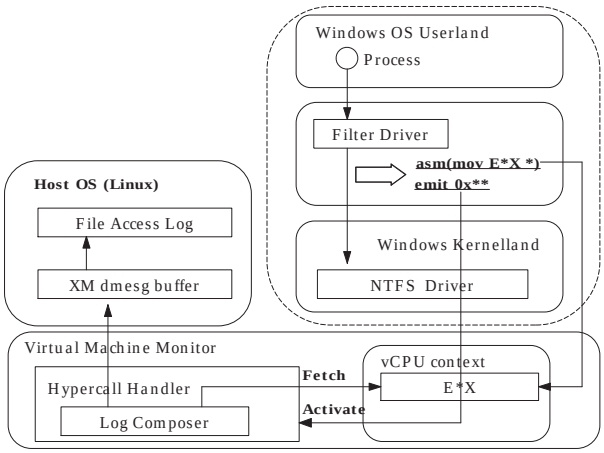


図 4 Xen における実装方式
Fig. 4 Implementation on Xen

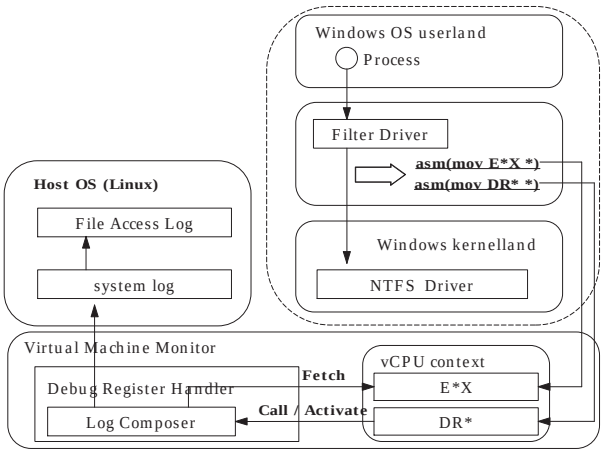


図 5 KVM における実装方式
Fig. 5 Implementation on KVM

4.3 仮想マシンモニタ側の修正

本節では，Xen と KVM における提案手法の実装を述べる．仮想マシンモニタ側では，ゲスト OS のレジスタ操作により呼び出されるハンドラを修正した．**図 4** と**図 5** で概要を示すが，Windows OS はクローズドソースのため，ソースコードを実装または修正できる Filter Driver と別に分けて記載してある．提案システムで実装を行うレイヤはフィルタドライバのレイヤである．

4.3.1 Xen に対する実装

図 4 に，Xen における提案手法の実装を示す．Windows OS のフィルタドライバは，デバッグレジスタに代入操作を行うことで，VM_EXIT を発生させ，Xen 内部のハイパーコールハンドラを起動させることができる．Xen は KVM のように Linux 内部のカーネルモジュールとして実装されていないため，提案システムでは，ハイパーコールハンドラ do_hvm_hypercall を修正し，仮想マシンモニタ内メッセージバッファを用いてホスト OS(domain0) にログを出力するように修正した．ログ文字列は Xen のメッセージバッファを経由して出力される．

```
int hvm_do_hypercall
(struct cpu_user_regs *regs)
```

ハイパーコールハンドラ do_hvm_hypercall は，hvm.c 内に実装されており，ゲスト OS 内でデバッグレジスタに代入操作を行うことで VM_EXIT が発生し，このハンドラに制御が移る．このハンドラは，vCPU という構造体を引数にもっているため，この仮想 CPU の引数であるレジスタにゲスト OS で数値を代入することで，仮想マシンモニタ側に値を通知することが可能である．このように，ゲスト OS 側で任意の時点でデバッグレジスタを操作し，汎用レジスタの値を通じて Xen 側にアクセスログについての情報を通知することができる．

4.3.2 KVM に対する実装

図 5 に，KVM における提案手法の実装を示す．提案手法では，仮想マシンモニタ内部のデバッグレジスタハンドラを用いて，ゲスト Windows OS のフィルタドライバとの通信を行っている．

フィルタドライバはログ文字列を分割し，vCPU の構造体を用いてドメイン間でログを転送する．ログを転送するために，Windows OS のフィルタドライバで，汎用レジスタに文字列とメッセージを格納し，デバッグレジスタ操作により，VM_EXIT を発生させ，仮想マシンモニタ内のデバッグレジスタハンドラを起動させる．x86 アーキテクチャでは，デバッグ用にレジスタが DR0 から DR6 まで用意されている．仮想マシンモニタ側は，dr_handler を修正し，デバッグレジスタが修正された際の汎用レジスタの値から，ゲスト OS からのログ情報を格納し，転送終了メッセージが受信された段階でアクセスログ文字列を再構築する．

5. 性能評価と考察

本章では，提案手法の有効性を評価するために，提案手法を Xen と KVM の双方で実装し，負荷を測定した結果とその考察を示す．評価実験ではマイクロベンチマークとして，ファイルアクセス観測時の負荷率を測定した．

5.1 マルウェアの検出結果の例

下記にマルウェアの検出結果の例を示す．blaster や sassar 等が用いる avserve2.exe が，Windows OS の感染に成功し，外部ホストへの新たな攻撃のために Windows OS のネットワークの設定の変更を試みていることが検出できる．

```
May 3 09:00:25 localhost kernel: [log][vm1];
filerw;PreRead;136;processId;4032;avserve2.
exe;HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001
```

\Services\Tcpip\ParametersDomain

また，このような特定のファイルアクセスの頻度を単位時間あたりに集計することで，マルウェアによって引き起こされる Windows OS 内部の異常検知を行うことが可能である．

5.2 実験の概要と実験環境

本節では，提案手法をそれぞれ仮想化なしのシステム (ベアマシン)，KVM ベースのシステム，Xen ベースのシステムに適用し，ディスク I/O に関するリソース利用率を測定した．評価実験環境は，Intel(R) Core(TM)2 Duo CPU (1.6GHz)，3.40GB のメモリを搭載したマシン上で実装し，ディスク I/O は Windows OS にデフォルトでインストールされているパフォーマンスモニタを用いた．測定項目は，Logical/Physical I/O の全項目を観測し，比較が有意なものを採り上げた．また，ベンチマークとして，大規模でファイル数の多い Linux のソースコード (Linux-3.1.0.tar.gz) を解凍して生成された約 4 万ファイルを，提案手法を実装したシステムのディスク上に書き込むことで行った．

5.3 実験結果と考察

表 1 はフィルタドライバ解除/稼動時の 3 システムでの Disk I/O の性能測定結果を示したものである．測定単位時間は 3 秒ごとに実行速度と転送バイト数を測定した．測定時間は，3 システム共通で 1350 秒である．

ここで，Avg Disk Bytes/Transfer は，ディスクに対する書き込み処理の実行速度の単位時間あたりの平均である．また，Avg. Disk Bytes/Write は，ディスクへ書き込みの際の転送バイト数の平均である．この値はディスクの待ち時間が反映されるケースがあるが，大きなデータの書き込みのコミットの際には待ち時間の増加は少なくなる．Windows OS には Windows 7 に搭載可能である，ドライバの署名が不要であるなどの理由から，XP を用いた．また，KVM は，Linux-3.1.0 に対応したカーネルモジュールと qemu，Xen は 4.1.2 と paravirt kernel-3.1.0 をコンパイルして負荷測定を行った．

測定結果は，Write/Transfer の 2 項目をフィルタドライバ解除/稼動をベアマシン，KVM，Xen で観測したため，全部で 12 ケースになる．概ね KVM ベースのシステムは全ての測定項目において，ベアマシン，Xen での実装と比較して，良好なパフォーマンスを示している．ベアマシンと Xen の比較であるが，フィルタドライバ解除/稼動時のケース双方において，パフォーマンスに顕著な違いがでないことが明らかになった．この点においては，フィルタドライバを用いた提案手法の負荷は許容できると結論できる．

表 2 は，提案手法のフィルタドライバ稼動時の 3 システムでの Disk I/O 性能の低下率を示したものである．KVM ベースのシステムでは低下率は殆どないことが確認された．また，ベアマシンでは，フィルタドライバ稼動時の低下率がもっとも多いことが確認された．この点において，ベアマシンよりも仮想化環境の方が，提案手法を適用する効果が高いことが示されたと言える．

5.4 他の関連研究との比較

2.1 節で述べた通り，Garfinkel の研究 [1] は，VM 観測に関する研究の代表的なものである．当研究は，外部観測手法を採用しており，性能評価としてポーリングベースの方式で実験測定を行っている．さらに，スナップショットなどの外部観測手法により得られる情報を一定時間間隔で取得し，rootkit のインストールや実行などの不正アクセスを検出している．当研究の評価実験では，ポーリングの間隔と，検出モジュールの処理が終わるまでの時間を測定しており，ポーリングの間隔は 2 秒から 8 秒の間隔で測定しているが，外部観測のため，イベントの検出に 10 秒前後かかっている．これに対し，提案システムは，内部観測を行うため，イベントの検出と記録にかかる所要時間は，i) フィルタドライバがファイルアクセスを検出する時間，ii) VM_EXIT が発生完了する時間，iii) ハイパーバイザーがレジスタから文字列を再構築する時間の 3 つを併せても VM 内部のファイルアクセスの捕捉時間とほぼ変わらないマイクロ秒単位となる．

また，2.2 節で述べた Secure In-VM Monitoring[20] はハードウェア仮想化による内部観測手法を適用した代表的な研究である．Sharif らの方式は，ゲスト VM 内のカーネルコードを変更することで CR3 の変更などのメモリ操作を捕捉し，MSR(model specific register) などを用いてゲスト VM の状態をハイパーバイザー側に通知する手法で，ゲスト OS のアーキテクチャに即したデバイスドライバなどは用いていない．この評価実験では，ゲスト VM 上のプロセス生成の捕捉に必要な性能低下率の測定を行っており，論文によると，外部観測手法では 690%程度に増加する (7 倍前後になる) 性能低下率を，Sharif らの方式では 13.7%程度に抑えることに成功している．Sharif らの方式ではデバイスドライバを用いず，CR3 や MSR へのアクセスなどのレジスタレベルでのイベントから観測ログを構築するため，高粒度な観測が行える一方で，前述したとおり 13.7%程度の性能低下が生じる．これに対し，提案システムでは専用のフィルタドライバを用いているため，表 2 で示したとおり，性能低下率は，KVM では 2%以下，Xen では 8%以下に留まっており，提案手法は Sharif らの方式より低い性能低下で外部観測を行うことができる．

5.5 マルウェアの検出率についての考察

現在，Windows OS だけを対象にしてもマルウェアの種類は数百万あるため，すべてのマルウェアに対して提案シ

表 1 フィルタドライバ解除/稼動時の 3 システムでの Disk I/O の性能測定結果
Table 1 Performance Measurements of Disk I/O with/without Filter Driver

item	filter	bare	KVM	Xen
Avg Disk Bytes/Transfer	no driver	4,230,333.138	6,501,465.529	3,693,278.987
Avg Disk Bytes/Transfer	driver	3,590,614.773	6,439,332.445	3,540,554.902
Avg Disk Bytes/Write	no driver	4,207,609.849	7,371,801.873	3,789,161.979
Avg Disk Bytes/Write	driver	3,568,011.354	7,274,390.209	3,501,165.831

表 2 フィルタドライバ稼動時の 3 システムでの Disk I/O 性能の低下率
Table 2 Decreasing Rate in Disk I/O Performance with Filter Driver

item	bare	KVM	Xen
Avg Disk Bytes/Transfer	0.8497	0.99	0.9586
Avg Disk Bytes/Write	0.8497	0.9867	0.9218

ステムの検出率を検討することは困難であるが，ある種のワームと，ファイルの書き換えによるレジストリの改ざんによりシステムへの常駐化を試みるトロイの木馬に限れば，提案システムは高い検出率を達成すると考察される．

すなわち，Microsoft 社のレポート [24] によれば，「ワーム」,「バックドア」, および「望ましくない可能性があるその他の悪質なソフトウェア」に関するカテゴリごとの年別推移の内，2011 年時点ではバックドアの割合が 5%，ワームの割合が 15%–20%である．代表的なワームは Win32/Msblast や Win32/Sassar であるが，いずれも提案システムで検出済みであり，同様に，代表的なバックドアは Win32/Rbot や Win32/Sdbot であるが，これらは通常システムへの常駐化を試みるためにレジストリの改ざんを行うため，提案システムによるファイルアクセスの捕捉によって検出可能であると推定される．一方で，メモリ操作経由で感染する rootkit やオンメモリのマルウェアは，提案システムでは検出が困難であるが，マルウェア全体に占める割合としては低い．

マルウェア検出の代表的な手法としては，メモリへのアクセスを監視する手法と，ファイルへのアクセスを監視する手法があるが，メモリへのアクセスを監視する手法は，主にメモリ内のシステムコールテーブル等を監視するので，ファイルやレジストリを改変するマルウェアを検知できない．一方で，ファイルへのアクセスを監視する手法は，メモリ操作経由で感染する rootkit やオンメモリのマルウェアをできない．この 2 つの手法は互いに異なるものを検知するものであり，2 つの方式を併用することで，より多くのマルウェアを検知することが可能になる．

5.6 提案システムの検出可能性と他の適用例について

VM 観測系の構築に関する重要な検討事項の 1 つに，攻撃側からの観測システムの検出可能性がある．一般に，デバッガなどのモニタリング専用の API を用いる内部観測は，外部観測と比較して検出される可能性が高いとされるが，この点において，提案システムでは，ファイルシステムドライバを用いているため，デバッガ API を用いたメモリ観測手法に比べて検出は困難である．特に，ファイルシステムドライバが用いる API は通常のファイルアクセスが用いるものと同じため，マルウェア側がこれを判別するのは不可能に近い．また，タイムスタンプなどを用いて，観測系が介在することによる処理速度の低下から検出を行う手法があるが，フィルタドライバは軽量に稼動するように設計されているため，マルウェア側からシステムが重いという点から異常検知することは難しい．また，VM 検出に関しては，外部観測も内部観測も検出可能性という点では同じであり，内部観測が外部観測と比べ検出可能性について不利であるということはない．

また，提案システムの適用として，マルウェア検知の他にフォレンジクスがある．フォレンジクスではファイルアクセスのロギングが重要になるため，専用の高粒度なファイルシステムのフィルタドライバを用いる提案システムの適用は有効である．特に，提案システムは Microsoft 社が提供しているフィルタリングマネージャを用いたデバイスドライバを実装しているため，従来の API フックやデバイスアクセスを捕捉する手法と比較して，軽量に動作することが可能であり，大量のログを取得処理しなければいけないフォレンジクスへの適用では有利である．通常，フォレンジクスではブロックデバイスのアクセスや外部からのシステムコール発行の捕捉を行い解析するため，大量の入出力要求が発生すること，さらにこれらを解析する場合，セマンティックギャップの問題からファイルアクセスに関して充分な情報が得られない場合があるが，提案システムでは，ファイルアクセスに関しては充分なログが取得可能である．また，Microsoft 社が提供するフィルタリングに特化したアーキテクチャを利用しているため，他の手法と比べて処理するべき I/O に関するイベントの量を少なく抑えることが可能である．

6. 結論

本論文では，ファイルシステムフィルタドライバを用いた内部観測による高粒度なドメイン間通信機構の実装方法を提案した．Windows OS に直接組み込まれたフィルタドライバからの通信により，仮想マシンモニタは外部観測と比較して多くの情報を得ることができる．しかしながら，このような欠点があるにも関わらず，現況のアプローチの多くはゲスト OS のメモリアクセスの外部観測方法を採用しており，この場合，ファイルアクセスなどの高レベルの API を捕捉する事は難しくなる．そのため，提案手法ではファイルアクセスをフックするためのフィルタドライバを開発した．

従来のゲスト OS のメモリ観測は，解析の高粒度化を可能にするが，メモリ観測だけでは補足出来ないトロイの木馬やアドウェア，スパイウェアなどの扱いに関しては有効な手段となっておらず，仮想マシン (特に Windows OS) のファイルアクセスを観測し，マルウェアの検出に必要な時系列データを生成する手法は提案されていない．前述したが，これらの検出にはファイルアクセスの観測が重要であり，提案システムはファイルアクセスとメモリアクセスの観測をあわせて行うことで，検出精度の向上が可能となる．従来のメモリのイントロスペクションによる手法は，OS の中でも比較的低レイヤで実行されるマルウェアに対する解析自体の高粒度化を主眼とするもので，マルウェア中でも大多数でありながら甚大な被害を及ぼすトロイの木馬やアドウェア，スパイウェア等の検出に対しては困難であるか，非効率な場合が多い．提案手法ではメモリアクセスと併せたファイルアクセスの詳細なイベントログを捕捉することにより，限られた解析自体の精度の向上だけでなく，マルウェア全体の検出範囲の拡大を可能にした．

既存のファイルアクセスを補足するソフトウェアは，ユーザ空間にて稼動するものであり，このため，システムへの高負荷やマルウェアによる実行の抑止などが行われる可能性が高かった．これに対し，提案システムはユーザ空間より堅牢なカーネル空間で実行されているファイルシステムのドライバを用いているため，安全かつ詳細なファイルアクセスのログを取得することが可能である．また，従来マルウェアなどの不正アクセスによるログはユーザ空間で実行されるプログラムによって保存されることが多く，そのためログの改ざんや消去，隠蔽が容易に行われてしまう．提案システムでは，ファイルシステムのドライバがロードされているカーネル空間でアクセスイベントを補足し，カーネル空間からハイパーバイザーへ直接ログの転送を行うため，安全かつ確実に不正アクセスのログを保存し，検証を行うことが可能である．

今後は，まず，第 2.3 節で示したような提案手法の効果について実証する方式について検討する．また，提案手法を用いて実際に異なるゲスト OS・仮想マシンモニタが混合する情報システムを構成し，仮想化環境において，複数システムのファイルアクセス情報を安全・確実に集約管理する方式についても検討することを計画している．

謝辞 貴重なご助言を頂いた編集委員と匿名査読者の方々に感謝いたします．

参考文献

[1] T. GARFINKEL. A virtual machine introspection based architecture for intrusion detection. In *Proc. of Network and Distributed Systems Security Symposium, 2003*, pp. 191–206, 2003.

[2] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, Vol. 37, pp. 164–177, October 2003.

[3] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the linux virtual machine monitor. In *Proceedings of the Linux Symposium*, Vol. 1, pp. 225–230, 2007.

[4] Carl A. Waldspurger. Memory resource management in vmware esx server. *SIGOPS Oper. Syst. Rev.*, Vol. 36, pp. 181–194, December 2002.

[5] Microsoft Corporation. Microsoft windows. http://www.microsoft.com/windows/.

[6] B.D. Payne. Xenaccess: An introspection library for xen, 2006.

[7] R. Sailer, E. Valdez, T. Jaeger, R. Perez, L. Van Doorn, J.L. Griffin, and S. Berger. shype: Secure hypervisor approach to trusted virtualized systems. *IBM Research Report RC23511*, 2005.

[8] D. Ferraiolo, J. Cugini, and D.R. Kuhn. Role-based access control (RBAC): Features and motivations. In *Proceedings of 11th Annual Computer Security Application Conference*, pp. 241–48, 1995.

[9] D. Ferraiolo and R. Kuhn. Role-Based Access Control. In *In 15th NIST-NCSC National Computer Security Conference*, 1992.

[10] J Tidswell and J Potter. An approach to dynamic domain and type enforcement. *Lecture Notes in Computer Science*, Vol. 1270, pp. 26–37, 1997.

[11] D.E. Bell and L.J. LaPadula. Secure computer systems: mathematical foundations and model. Technical Report M74-244, The MITRE Corporation, Bedford, MA, 1973.

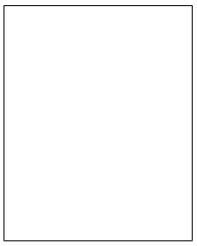
[12] K. J. Biba. Integrity considerations for secure computer systems. Technical Report MTR-3153, The MITRE Corporation, Bedford, MA, 1977.

[13] DD Clark and DR Wilson. A comparison of commercial and military computer security models. In *Proceedings of the IEEE Computer Society Symposium on Security and Privacy*, pp. 184–194. IEEE Computer Society, 1987.

[14] Volatile Systems, LLC. Volatility — memory forensics — volatile systems. https://www.volatilesystems.com/default/volatility.

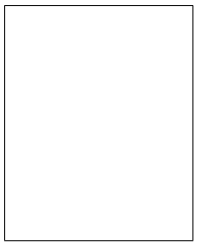
[15] George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza A. Basrai, and Peter M. Chen. Revirt: enabling intrusion analysis through virtual-machine logging and replay. *SIGOPS Oper. Syst. Rev.*, Vol. 36, pp. 211–224,

December 2002.
[16] B.D. Payne, M. Carbone, M. Sharif, and W. Lee. Lares: An architecture for secure active monitoring using virtualization. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pp. 233–247. IEEE, 2008.
[17] Nguyen Anh Quynh and Kuniyasu Suzaki. Xenprobes, a lightweight user-space probing framework for xen virtual machine. In *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, pp. 2:1–2:14, Berkeley, CA, USA, 2007. USENIX Association.
[18] Lee Breslau, Chris Chase, Nick Duffield, Bill Fenner, Yanhua Mao, and Subhabrata Sen. Vmscope: a virtual multicast vpn performance monitor. In *Proceedings of the 2006 SIGCOMM workshop on Internet network management*, INM '06, pp. 59–64, New York, NY, USA, 2006. ACM.
[19] Peter M. Chen and Brian D. Noble. When virtual is better than real. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, pp. 133–, Washington, DC, USA, 2001. IEEE Computer Society.
[20] Monirul I. Sharif, Wenke Lee, Weidong Cui, and Andrea Lanzi. Secure in-vm monitoring using hardware virtualization. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pp. 477–487, New York, NY, USA, 2009. ACM.
[21] L. Litty, H.A. Lagar-Cavilla, and D. Lie. Hypervisor support for identifying covertly executing binaries. In *Proceedings of the 17th conference on Security symposium*, pp. 243–258. USENIX Association, 2008.
[22] N.A. Quynh, R. Ando, and Y. Takefuji. Centralized security policy support for virtual machine. In *USENIX, 20th Large Installation System Administration Conference*, 2006.
[23] S.T. King, G.W. Dunlap, and P.M. Chen. Debugging operating systems with time-traveling virtual machines. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, pp. 1–1. USENIX Association, 2005.
[24] Microsoft Corporation. セキュリティインテリジェンスレポート.
http://www.microsoft.com/ja-jp/security/resources/sir.aspx.



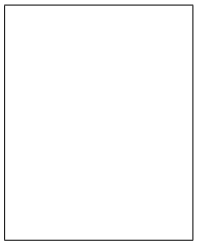
安藤 類央 （正会員）

2006 年，慶應義塾大学政策・メディア研究科後期博士課程修了，博士（政策・メディア）．同年情報通信研究機構所属．ネットワークセキュリティ，仮想化システムセキュリティの研究に従事．情報処理学会会員．



橋本 正樹 （正会員）

2010 年 3 月，情報セキュリティ大学院大学情報セキュリティ研究科修了，博士（情報学）．同年 4 月より情報セキュリティ大学院大学情報セキュリティ研究科・助教．情報処理学会，電子情報通信学会，日本ソフトウェア科学会，IEEE 各会員．電子情報通信学会 ISS・情報通信システムセキュリティ研究会専門委員．



山内 利宏 （正会員）

1998 年九州大学工学部情報工学科卒業．2000 年同大大学院システム情報科学研究科修士課程修了．2002 年同大学院システム情報科学府博士後期課程修了．2001 年日本学術振興会特別研究員（DC2）．2002 年九州大学大学院システム情報科学研究院助手．2005 年岡山大学大学院自然科学研究科助教授．現在，同准教授．博士（工学）．オペレーティングシステム，コンピュータセキュリティに興味を持つ．電子情報通信学会，ACM，USENIX 各会員．