

# KVMにおける仮想マシンの内部監視機構の実装と性能評価

中村孝介<sup>1</sup> 光来健一<sup>1,2</sup>

**概要:** 本稿では、Linux KVMにおいて、仮想マシン (VM) を用いた侵入検知システム (IDS) のオフロードを実現するシステム *KVMonitor* を提案する。*KVMonitor* は IDS が VM の外から安全にメモリやディスク、ネットワークの監視を行うことを可能にする。我々は *KVMonitor* を用いて、既存の IDS をオフロードするための実行環境を提供する *Transcall* を KVM に移植した。*Transcall* を用いた実験により、オフロードした IDS が正しく攻撃を検出できることを確認した。また、KVM と Xen における IDS オフロードの性能比較を行った。

## Implementation and Evaluation of Virtual Machine Introspection in KVM

KOUSUKE NAKAMURA<sup>1</sup> KENICHI KOURAI<sup>1,2</sup>

**Abstract:** In this paper, we propose the system for offloading intrusion detection systems (IDSes) with virtual machines (VMs) in Linux KVM, called *KVMonitor*. *KVMonitor* enables IDSes to securely monitor the memory, disks, and networks of VMs from the outside. Using *KVMonitor*, we have ported *Transcall*, which provides execution environments for offloading the existing IDSes, to KVM. We confirmed that offloaded IDSes could detect attacks correctly. In addition, we compared the performance of IDS offloading in KVM and Xen.

### 1. はじめに

インターネットに接続されたサーバへの攻撃は年々増加している。攻撃を受けると、重要な情報を盗まれたり、サーバの機能を停止させられてサービスが提供できない状態になったりする。このような攻撃者の侵入を検知する手段として侵入検知システム (IDS) が用いられている。IDS はサーバのシステムやネットワークの監視を行い、異常を検知したら管理者に通知を行う。しかし、近年、攻撃者は IDS に検知されるのを防ぐために、侵入後に IDS の無効化や改ざんを試みるようになってきた。IDS が攻撃を受けるとそれ以降の攻撃を検知できなくなってしまう。

このような IDS 自身への攻撃に対処するために、仮想マシン (VM) を用いて IDS をオフロードするという手法が提案されている [1]。IDS オフロードは監視対象システムを VM 上で動作させ、IDS を監視対象 VM の外側で実行して監視を行う手法である。IDS オフロードを行うことにより、監視対象 VM に侵入されたとしても IDS を攻撃することはできず、IDS のセキュリティを向上させることができる。

これまで、IDS オフロードの研究は様々な仮想環境で行われてきた [1], [2], [3] が、KVM における IDS オフロードの研究はほとんど行われていない。KVM は急速に普及しつつあるため、IDS オフロードを実現できるようにすることがセキュリティを向上させるために必要である。我々はこれまでに、Xen [4] を用いて様々な IDS オフロードの研究を行ってきた [5], [6], [7]。しかし、KVM のアーキテク

<sup>1</sup> 九州工業大学  
Kyushu Institute of Technology

<sup>2</sup> 独立行政法人科学技術振興機構 CREST  
JST, CREST

チャは Xen とは大きく異なるため、Xen における IDS オフロードの手法を適用できるかどうかは不明であった。また、異なるアーキテクチャ間での IDS オフロードの性能比較も行われてこなかった。

本稿では、KVM において IDS オフロードを実現するシステムである KVMonitor を提案する。KVMonitor は IDS をホスト OS 上にオフロードし、VM のメモリやディスク、ネットワークの監視を行うことを可能にする。IDS が VM のメモリを監視できるようにするために、VM に割り当てる物理メモリをファイルとして作成し、VM と IDS の両方にメモリマップする。ディスクの監視のために、ネットワーク・ブロックデバイスを用いて、KVM の標準形式のディスクイメージを扱えるようにする。また、IDS オフロードに伴って生じる VM の性能分離の問題を解決するために、ホスト OS の Cgroups 機構を用いて VM とオフロードした IDS をグループ化する。

我々は KVMonitor を QEMU-KVM 1.1.2 に対して実装し、既存の IDS をオフロード可能にする Transcall [7] を KVM に移植した。KVMonitor を用いていくつかの IDS を実行することにより、ホスト OS 上の IDS が VM 内の異常を検知できることを確かめた。さらに、Xen における IDS オフロードとの性能比較を行った。基本的な性能として、VM に対するメモリ読み込み性能、ディスク読み込み性能、パケットキャプチャ性能を調べ、IDS の実行時間の比較も行った。

以下、2 章で Xen における IDS オフロードについて説明し、3 章で KVMonitor について述べる。4 章では KVMonitor を用いて行った実験について述べる。5 章で関連研究に触れ、6 章で本稿をまとめる。

## 2. 背景

### 2.1 Xen における IDS オフロード

Xen においては、IDS をドメイン U からドメイン 0 にオフロードするのが一般的である。ドメイン 0 はドメイン U を管理するための特権を持った VM であり、ドメイン U 内の様々なリソースを監視することが可能である。ドメイン 0 以外の VM にオフロードする手法として、ドメイン M [8] や SSC [9] などが提案されており、仮想マシンモニタ (VMM) にオフロードすることも可能であるが、本稿ではドメイン 0 にオフロードする場合について考える。

ドメイン 0 にオフロードした IDS は、VMM の機能を用いてドメイン U のメモリをマップすることでメモリの監視を行う。プロセス構造体など、特定のカーネルデータを監視するには、VM イントロスペクション [1] と呼ばれる技術を用いる。VM イントロスペクションはアドレス変換を行ってカーネルデータの仮想アドレスからそのデータが置かれている物理ページを特定し、そのページをマップすることでアクセスを行う。物理ページを特定するにはま

ず、ハイパーコールを発行して VMM からドメイン U の仮想 CPU の CR3 レジスタの値を取得する。次に、CR3 レジスタに格納されているページディレクトリアドレスを起点として、ページテーブルが格納されている物理ページのマップを数回繰り返してページテーブルをたどる。

ドメイン 0 の IDS は、ドメイン 0 上に置かれているドメイン U のディスクイメージにアクセスすることで、ディスクの監視を行う。Xen では、ドメイン U のディスクイメージはドメイン 0 のファイルまたは、ディスク・パーティションとして作成される。ファイルとして作成されたディスクイメージには、デフォルトで raw 形式が用いられており、ドメイン 0 のカーネルがファイルシステムをサポートしていれば、そのままループバック・マウントすることができる。論理ボリュームマネージャ (LVM) が用いられている場合には、論理ボリュームをアクティブにしてからマウントする。このディスクイメージはドメイン U からマウントされているため、ファイルシステムを破壊しないように読み込み専用でマウントする。

ドメイン 0 の IDS はドメイン U のネットワーク・インタフェースからパケットをキャプチャすることで、ネットワークの監視を行う。Xen では、ドメイン U のネットワーク・インタフェースに対応するバックエンドが仮想インタフェース (vif) としてドメイン 0 に作成される。ドメイン 0 がドメイン U 宛てのパケットを受信すると、ネットワーク・ブリッジと仮想インタフェースを経由して、ドメイン U のネットワーク・インタフェースにパケットが送られる。逆に、ドメイン U がパケットを送信すると、ドメイン 0 の仮想インタフェースとネットワーク・ブリッジを経由して外部に送信される。そのため、ドメイン 0 の仮想インタフェースでドメイン U が送受信するすべてのパケットをキャプチャすることができる。

### 2.2 IDS オフロード時の性能分離

VMM は VM 単位でリソース管理を行っているため、ドメイン 0 に IDS をオフロードすると IDS はドメイン 0 のリソースを消費することになる。ドメイン 0 はすべてのドメイン U で共有されているため、IDS オフロードによってドメイン 0 のリソースが消費されるとドメイン U 間の公平が保てなくなる。例えば、大量のリソースを消費する IDS をオフロードしたドメイン U はその分だけ多くのリソースを利用できてしまう。オフロードする前の IDS はドメイン U のリソースを消費していたため、このような不公平は生じなかった。

この問題を解決するために、Resource Cage と呼ばれるリソース管理手法が提案されている [5]。Resource Cage は VM とオフロードした IDS をひとまとまりとして管理することを可能にしている。OffloadCage [5] は Xen のクレジットスケジューラに変更を加えて、IDS プロセスが消費

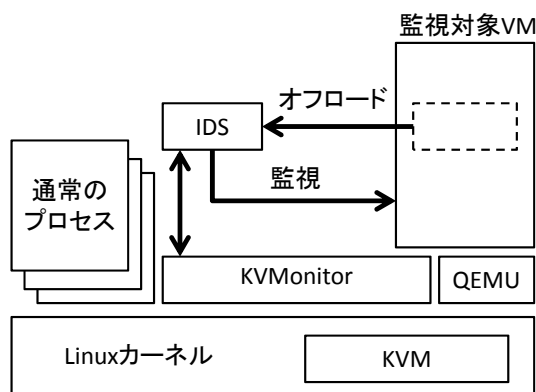


図1 KVMonitorの構成

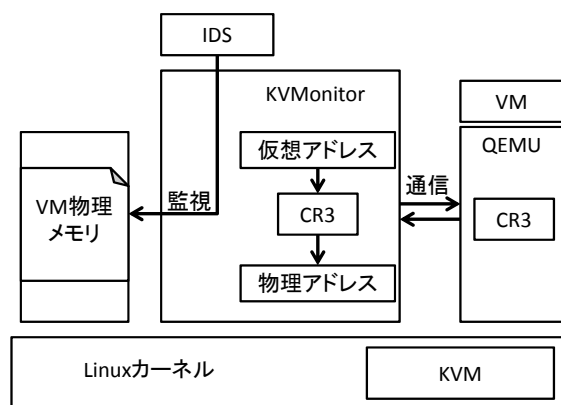


図2 メモリアドレスの変換

したCPU時間を考慮してドメインUに配分するCPU時間を減らす。Balloon Performer [6]はIDSプロセスが消費したメモリ量をドメインUへのメモリ割り当てから減らすことで、合計のメモリ量を一定に保つ。IDSプロセスの消費メモリには、ファイルを読み込むことによってOS内に作られるファイルキャッシュも考慮されている。しかし、ドメイン0のIDSはOSによって管理されているのに対し、ドメインUはVMMによって管理されているため、IDSとドメインUの間でのリソース割り当てを柔軟に行うのは容易ではない。

### 3. KVMonitor

本稿では、KVMにおいてIDSオフロードを実現するシステムであるKVMonitorを提案する。KVMonitorはVMのメモリとディスク、ネットワークの監視に対応している。

#### 3.1 アーキテクチャ

KVMonitorは図1のように、IDSをホストOS上のプロセスとしてオフロードする。KVMはLinuxカーネルの中でVMMを動作させ、VMをホストOSの一つのプロセスとして管理している。このプロセスはQEMUと呼ばれるエミュレータであり、VMにメモリ、ディスク、ネットワークなどの仮想デバイスを提供している。ホストOS上のIDSはQEMUが管理するこれらの仮想デバイスにアクセスすることで、VMの監視を行うことができる。Xenと同様に、監視用のVMを作ってIDSをオフロードするアーキテクチャも考えられるが、KVMには特権を持ったVMが存在しないため、ホストOSにアクセスできるVMが必要になる。そのため、ホストOS上に直接、IDSをオフロードするアーキテクチャを採用した。

#### 3.2 メモリの監視

オフロードしたIDSがVMのメモリを監視できるようにするために、KVMonitorはVMに割り当てる物理メモリをホストOS上のファイル（メモリファイル）として作成する。このファイルをQEMUとIDSの両方にマップす

ることにより、VMに対しては従来通りの直接メモリアccessを可能にしつつ、外部のIDSもそのメモリを参照することができる。従来のQEMUではVMの物理メモリをmallocによって確保していたため、QEMUの外部から参照することはできなかった。QEMUは元々、ファイルをVMの物理メモリとして用いるオプションを提供していたが、他のプロセスがこのファイルにアクセスできないようにファイルを削除していた。そこで、QEMUに修正を加えてファイルを削除しないようにした。

IDSは仮想アドレスを用いてVMのメモリ上のデータにアクセスするため、KVMonitorが仮想アドレスを物理アドレスに変換する。メモリマップされるVMのメモリは物理メモリであり、物理アドレスでアクセスするためである。KVMonitorはメモリアドレスを変換するために必要なCR3レジスタの値をQEMUと通信して取得する。既存のQEMUは外部からCR3レジスタの値を取得する手段を提供していなかったため、CR3レジスタの値を返すコマンドをQEMUに追加した。KVMonitorはQEMU Monitor Protocol (QMP) を用いてQEMUのコマンドを実行する。KVMonitorは取得したCR3レジスタの値を用いてページテーブルをたどり、アドレス変換を行う。

QEMUはHugeTLBファイルシステム(hugetlbfs)をマウントしたディレクトリにメモリファイルを配置することで、ホストOSのメモリ性能への影響を抑える。HugeTLBファイルシステムはファイルをメモリマップする際に、ページサイズを4KBから2MBに拡張するHugeページ機構を用いる。ページサイズを大きくすることにより、TLBのエントリの消費を抑え、アドレス変換も高速に行うことができるようになる。

#### 3.3 ディスクの監視

KVMのデフォルトであるqcow2形式のディスクイメージを監視できるようにするために、KVMonitorはネットワーク・ブロックデバイス(NBD)の機能を用いる。qcow2形式はディスクに割り当てたサイズではなく、実際に使用

しているサイズのディスクイメージを作成するため、ディスクスペースの節約ができるという利点を持っている。しかし、qcow2形式のディスクイメージはホストOS上で直接マウントすることができない。そこで、KVMonitorではNBDの機能を用いてqcow2形式のディスクイメージを仮想的なブロックデバイスとしてマウントすることで、IDSにディスクを監視する環境を提供する。

KVMonitorはqcow2形式のディスクイメージをqemu-nbdと呼ばれるツールを用いてホストOS上にマウントする。qemu-nbdはNBDサーバとして振る舞い、qcow2形式などの特殊なディスクイメージを仮想的なブロックデバイスとしてnbd-clientに見せる。nbd-clientはそのブロックデバイスをブロックデバイス・ファイルに結びつける。KVMonitorはこのブロックデバイス・ファイルをマウントし、IDSがアクセスするたびにqemu-nbdと通信してディスクブロックを変換する。ファイルシステムの整合性を保つために、KVMonitorはVMのディスクイメージを読み込み専用でマウントする。

### 3.4 ネットワークの監視

KVMonitorはVMをホストOSのネットワークにブリッジ接続し、QEMUが作成するtapデバイスをIDSに提供する。tapデバイスは、ホストOSがQEMUと通信するための仮想的なイーサネットデバイスを提供する機能である。KVMのデフォルトのネットワーク構成はNAT接続になっているが、この場合には、VM外部から監視できるインタフェースは作成されないため監視を行うことができない。

### 3.5 Transcallの移植

Transcall [7]はVM Shadowと呼ばれる実行環境を提供し、その中で動作するIDSがVMを透過的に監視することを可能にする。Transcallを用いることで、既存のIDSを修正することなくオフロードして実行することができるようになる。Transcallはシステムコール・エミュレータとShadowファイルシステムで構成されている。システムコール・エミュレータはIDSが発行するシステムコールにVM内の情報を返させるために、VM内のカーネルデータから必要な情報を取得する。ShadowファイルシステムはVM内と同一のファイルシステムを提供しつつ、安全のためにIDSの実行に必要なファイルだけホストOS上のファイルを使わせる。Shadowファイルシステムの一部であるShadow procファイルシステムは、VM内のメモリを解析して、プロセスやネットワークに関する情報を提供する。

我々はXenを対象として開発されていたTranscallをKVMに移植した。そのために、Transcallのシステムコール・エミュレータとShadow procファイルシステムについて、KVMonitorを用いてVMのメモリにアクセスするよ

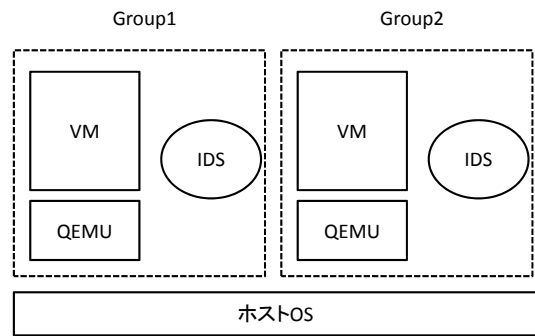


図3 VMとIDSのグループ化

うにした。

### 3.6 VMの性能分離

オフロードしたIDSとVMをひとまとまりとしたリソース管理を行えるようにするために、KVMonitorはLinuxのCgroups機能を用いる。Cgroupsは複数のプロセスをグループ化して、そのグループ単位でリソース割り当てを行う機能である。この機能を用いて図3のようにホストOS上でIDSとVMの2つのプロセスをグループ化する。このグループに対してCPUやメモリへの制約を設定することで、グループ間での性能分離を実現する。このようなグループ化はKVMにおいて、VMがホストOSの1つのプロセスとして作成されているために容易に行うことができる。

グループに対するCPUに関する制約として、シェアと上限を設定することができる。シェアにはグループ間でのCPU時間の相対的配分を指定することができる。上限は100msの間にCPUを割り当てる最大時間を指定することで設定する。これらのCPU時間はIDSとVMの間で任意に配分される。一方、メモリに関する制約として、グループに割り当てるメモリ量の上限を設定することができる。このメモリ量にはホストOS内に確保されたファイルキャッシュも含まれるため、ディスクの監視のようにファイルキャッシュを大量に使うIDSについても、使用するメモリ量を厳密に制限することができる。割り当てられたメモリ量はIDSとVMの間で任意に配分される。

## 4. 実験

KVMonitorを用いてオフロードされたIDSが正常に監視を行えることを確認する実験を行った。また、KVMとXenにおいてオフロードされたIDSの監視性能を比較する実験も行った。KVM用とXen用にそれぞれ、Intel Xeon E5630 (4コア、2.53GHz、L3キャッシュ12MB)のCPU、6GBのメモリ(DDR3、1066MHz)、250GBのHDDを搭載したマシンを用いた。KVM用のマシンでは、修正を加えたQEMU-KVM 1.1.2を用い、ホストOSとしてLinux 3.2.0を動作させた。Xen用のマシンでは、Xen 4.1.3を用い、ド

メイン 0 の OS として完全仮想化で動作する Linux 3.2.0 を動作させた。VM 内のゲスト OS には Linux 2.6.27.35 を用いた。

#### 4.1 動作テスト

##### 4.1.1 IDS オフロード

オフロードした chkrootkit [10] が正しくルートキットを検出できることを確認するために、ホスト OS 上で Trascall を用いて chkrootkit を実行した。VM 内の ps コマンドと netstat コマンドを改ざんしたところ、ホスト OS 上からでも chkrootkit は改ざんを検出することができた。

次に、ホスト OS 上で Tripwire [11] を実行し、VM のディスクの監視を行った。Tripwire のポリシーファイルには VM の /home 以下のファイルを監視するように記述した。まず、ホスト OS 上で Tripwire を実行してファイルの正常な状態を記録し、VM 内で 3 つのファイルについて追加、削除、変更を行った。その後で、ホスト OS 上で Tripwire を実行してファイルの検査を行ったところ、この 3 つのファイルの変化を正しく検出することができた。

最後に、Snort [12] をホスト OS 上にオフロードし、ポートスキャンを検出できることを確認する実験を行った。Snort には VM の tap デバイスを監視するように指定した。ポートスキャンを行うために、VM に対して nmap コマンドを実行した。その結果、Snort の警告ログがホスト OS 上の /var/log/snort/alert に正常に記録された。

##### 4.1.2 Cross-View Diff

隠しプロセスを発見するために、ホスト OS にオフロードした ps コマンドと VM 上の ps コマンドの実行結果を比較した。オフロードした ps コマンドは Trascall を用いて実行した。この実験では、ルートキットを模倣して VM 上の ps コマンドを改ざんし、init プロセスを表示しないようにした。オフロードした ps コマンドは正常に init プロセスを表示したため、VM 上の ps コマンドの実行結果と比較することにより、VM 上で隠されている init プロセスを見つけることができた。

次に、ネットワークの隠しポートを発見するために、ホスト OS にオフロードした netstat コマンドと VM 上の netstat コマンドの実行結果を比較した。この実験では、5900 番ポートを表示しないように VM 上の netstat コマンドを改ざんした。これらのコマンドの実行結果を比較したところ、オフロードした netstat コマンドだけが 5900 番ポートを表示したため、VM 上の netstat コマンドがこのポートを隠していることを検出することができた。

#### 4.2 監視性能の比較

KVM と Xen における監視性能を比較するために、以下の 4 つの場合について実験を行った。

- **KVM\_host** KVM のホスト OS から VM を監視

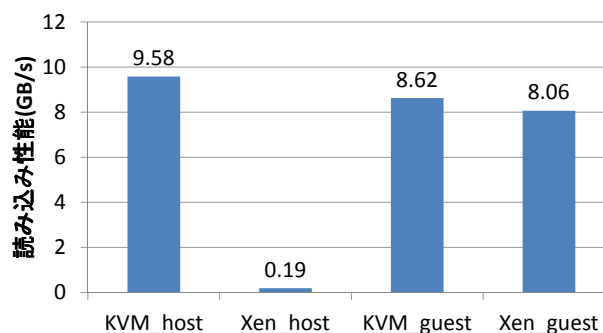


図 4 メモリ読み込み性能

- **Xen\_host** Xen のドメイン 0 から VM を監視
- **KVM\_guest** KVM の VM 内で自身を監視
- **Xen\_guest** Xen のドメイン U 内で自身を監視

##### 4.2.1 メモリ監視性能

メモリ監視性能の比較を行うために、VM のメモリを読み込むベンチマークを行った。このベンチマークは VM のメモリをページ単位でアクセスし、確保した 4 KB の領域に memcpy 関数を用いてコピーした。この実験において、**KVM\_host** では KVM のホスト OS 上に VM のメモリファイルをマップしてアクセスし、**Xen\_host** では Xen のドメイン 0 に VM のメモリページをマップしながらアクセスした。この際にアドレス変換は行わず、直接、VM の物理メモリにアクセスを行った。**KVM\_guest** と **KVM\_host** については、VM 内で malloc を用いてメモリを確保し、一旦、書き込みを行った後でメモリの読み込み時間を測定した。

図 4 に示す実験結果より、KVM のホスト OS からの読み込み性能 (**KVM\_host**) は VM 内での性能 (**KVM\_guest**) よりも高いことが分かった。これは、KVMonitor がメモリファイルをマップした後は通常のメモリアクセスと同等であり、また、KVM のホスト OS には仮想化のオーバーヘッドがないためである。一方、Xen のドメイン 0 からの読み込み性能 (**Xen\_host**) は極端に遅いことが分かった。これは、ドメイン 0 はドメイン U のメモリをページ単位でマップしてアクセスし、その後でアンマップしなければならないためである。

次に、VM 内のカーネルのテキスト領域を読み出して検査を行う IDS を実行し、監視性能の比較を行った。この IDS はテキスト領域の先頭から最後までアドレスを 4 KB 単位で変換し、目的の物理メモリの内容を読み込む。アドレス変換を行うために VM のメモリ上のページテーブルにもアクセスする必要がある。実験結果を図 5 に示す。KVMonitor を用いた場合の **KVM\_host** は、Xen でオフロードを行った場合 (**Xen\_host**) の 100 倍以上高速であった。

##### 4.2.2 ディスク監視性能

ディスク監視性能の比較を行うために、IOZone ベンチ

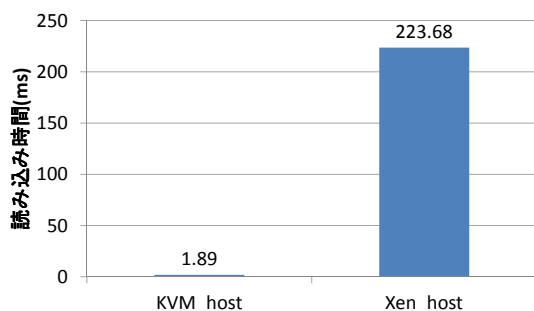


図 5 カーネル監視時間

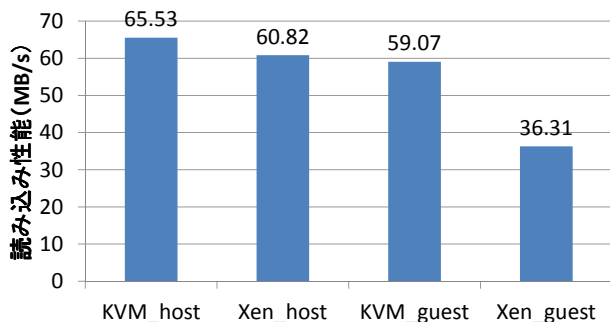


図 6 ディスク読み込み性能

マーク [13] を用いてファイル読み出し性能を調べた。この実験では、IOZone を用いて VM のディスク上に 1GB のテスト用ファイルを作成しておき、そのファイルを読み込む性能を測定した。このファイルは ext4 ファイルシステム上に作成した。KVM\_host と Xen\_host ではディスクイメージをマウントしてファイルへのアクセスを行い、KVM\_guest と Xen\_guest では直接、読み込みを行った。OS 内のキャッシュの影響を排除するために、測定を行うたびに Linux の機能を用いてキャッシュを消去した。

IOZone を 10 回実行した時の平均読み込み性能を図 6 に示す。実験結果から、KVM、Xen とともに VM 内でファイルを読み込む場合 (KVM\_guest、Xen\_guest) より、ホスト側で読み込む場合 (KVM\_host、Xen\_host) のほうが高速であることが分かった。KVM の場合は 11 % 高速になるだけであったが、Xen の場合は 68 % 高速になった。これは、VM のディスクイメージがホスト側に置かれており、ホスト OS は仮想化のオーバーヘッドなしでアクセスできるためと考えられる。KVM と Xen を比較すると、ホスト OS とドメイン 0 からの読み込み性能の差は小さいが、ドメイン U 内での読み込み性能は KVM の VM 内での性能の 61 % であることが分かった。

次に、ディスクイメージの形式の違いによる監視性能への影響を調べるために、KVM と Xen でそれぞれ raw 形式、qcow2 形式を用いた場合についても測定を行った。図 7 にすべての組み合わせについての読み込み性能を示す。KVM の場合は、ディスクイメージの形式の違いによる影響はほぼないことが分かった。一方、Xen の場合は、qcow2 形式を用いるほうがドメイン 0 からの読み込みは 19 % 高速に

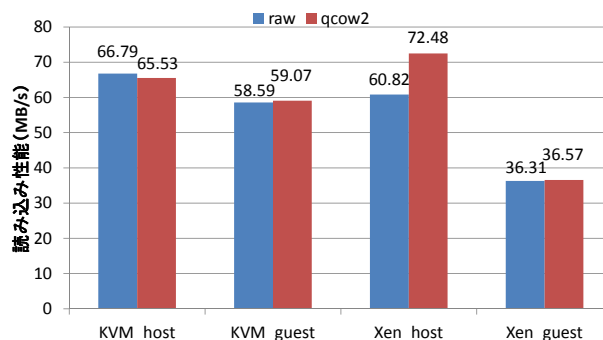


図 7 ディスクイメージ形式の読み込み性能への影響

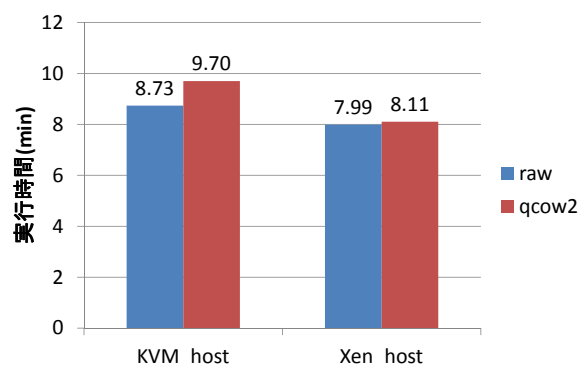


図 8 Tripwire の実行時間

なることが分かった。いずれの場合も、VM 内での読み込み性能への影響はほぼなかった。

#### 4.2.3 Tripwire の実行時間

オフロードした Tripwire を用いて、VM のディスクを監視するのにかかる時間を測定した。この実験も、KVM と Xen のディスクイメージに raw 形式と qcow2 形式の両方を用いた場合について行った。実験条件を統一するために、raw 形式のディスクイメージを qcow2 形式に変換し、KVM と Xen で同じ内容のディスクイメージを使用した。また、Tripwire のポリシーファイルには VM のディスク全体を監視するように記述した。

図 8 に実験結果を示す。どちらの形式のディスクイメージを用いた場合も、Xen における実行時間のほうが KVM より短いという結果となった。Xen と KVM のデフォルトである raw 形式、qcow2 形式をそれぞれ用いた場合には、Xen のほうが 18 % 高速であった。ディスクイメージの形式に関して比較すると、KVM、Xen とともに raw 形式における実行時間が qcow2 形式の場合より短くなった。特に、KVM では raw 形式に変更したほうが 10 % 高速になった。これは、qcow2 形式のディスクイメージはアクセスされるたびに NBD で変換されるため、ディスクアクセスのオーバーヘッドが顕著に現れることが原因と考えられる。

#### 4.2.4 パケットキャプチャ性能の比較

オフロードした Snort の性能を調べるために、VM に大量のパケットを送った時の Snort のパケットロス率を測定した。この実験では、トラフィックジェネレータの D-ITG

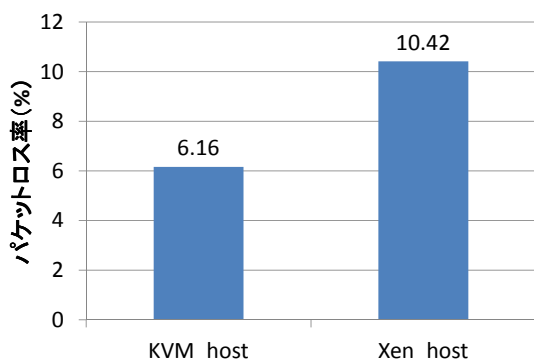


図 9 Snort のパケットロス率

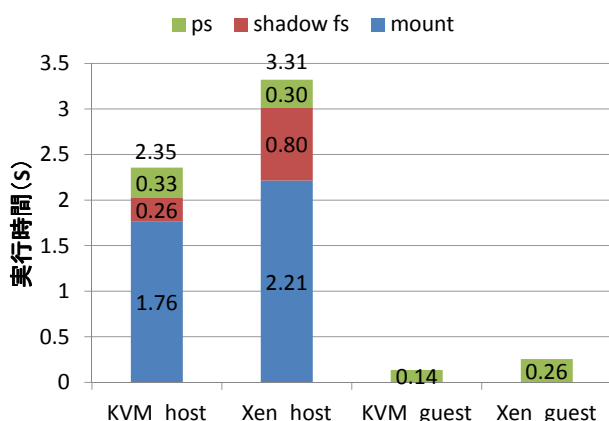


図 10 ps コマンドの実行時間

2.8.0-rc1 [14] を用いて VM のネットワークに負荷をかけた。VM 上のシステムが古く、D-ITG の受信側プログラムがコンパイルできなかったため、この実験では、ホスト側と同じシステムをインストールした VM を用いた。実験結果は図 9 のようになり、KVM のホスト OS 上で Snort を実行したほうが Xen のドメイン 0 で実行した場合よりパケットロス率は低いことが分かった。これはドメイン 0 も VM であるため仮想化が行われており、仮想化されていない KVM のホスト OS と比べると性能が劣るためと考えられる。

#### 4.2.5 プロセス情報の取得時間

ps コマンドをオフロードして実行した場合と、VM 内で実行した場合の実行時間を測定した。オフロードした ps コマンドは Trancall を用いて実行した。それぞれ 10 回測定を行った時の平均を図 10 に示す。実験結果から、KVM においてオフロードした場合のほうが Xen よりも 29% 高速であることが分かった。その内訳を見ると、Xen では Shadow proc ファイルシステムの作成とディスクイメージのマウントに時間がかかっているのが原因であることが分かる。一方、VM 内で実行した場合と比較すると、これらのオーバーヘッドのせいで非常に時間がかかっていることが分かる。

#### 4.2.6 chkrootkit の実行時間

chkrootkit をオフロードし、Trancall を用いて実行し

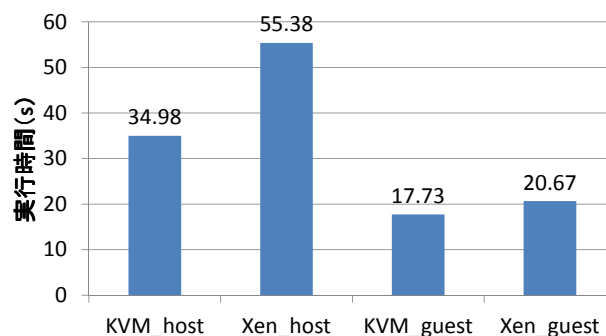


図 11 chkrootkit の実行時間

た場合と、VM 内で実行した場合の実行時間を測定した。chkrootkit はファイルの検査も行うため、OS 内のキャッシュの影響を排除するために、測定を行うたびにキャッシュを消去した。KVM と Xen ではそれぞれデフォルトの qcow2 形式、raw 形式のディスクイメージを用いた。それぞれ 10 回測定を行った時の平均を図 11 に示す。オフロードした時の実行時間には VM のディスクイメージのマウント・アンマウント、Shadow proc ファイルシステムの作成の時間も含まれている。

KVMonitor を用いて chkrootkit をオフロードした場合の実行時間 (KVM\_host) は VM 内で実行した場合 (KVM\_guest) の 2.0 倍となった。Xen ではオフロードすると 2.7 倍の実行時間がかかることから、KVM におけるオフロードのほうが性能低下の度合いは小さいことが分かった。

### 4.3 VM の性能分離

#### 4.3.1 CPU 割り当ての制約

VM と IDS からなるグループに割り当てる CPU 時間に制約をかけることができることを確かめる実験を行った。グループ 1 には VM とオフロードした Tripwire を含め、VM 上でループを行う loop プログラムを実行した。グループ 2 ではホスト OS 上で同じ loop プログラムを実行した。グループ 1 とグループ 2 の CPU シェアは 60 : 40 に設定した。これらのプログラムの実行を一定時間ごとに制御して、CPU 割り当てが正しく行われるかどうかを調べた。

図 12 は各プログラムとグループ 1 の CPU 使用率の変化を示している。前半では各グループに属しているプロセスを単独で実行し、後半ではグループ 1 の Tripwire とグループ 2 の loop プログラムを同時に実行した。プログラムを単独で実行した場合は、CPU シェアの設定に関わらずほぼ 100% の CPU を使用した。一方、2 つのグループ内のプログラムを同時に実行した時は CPU 使用率が 60% と 40% になった。

次に、(1) グループ 1 内の 2 つのプログラムを同時に実行した場合、(2) 2 つのグループ内の loop プログラムを同時に実行した場合、(3) 全プログラムを同時に実行した場

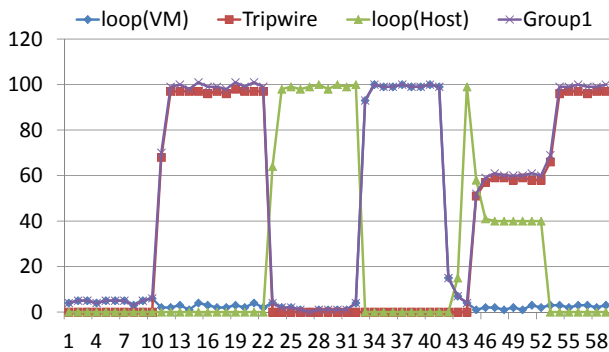


図 12 CPU 使用率の変化 (パターン 1)

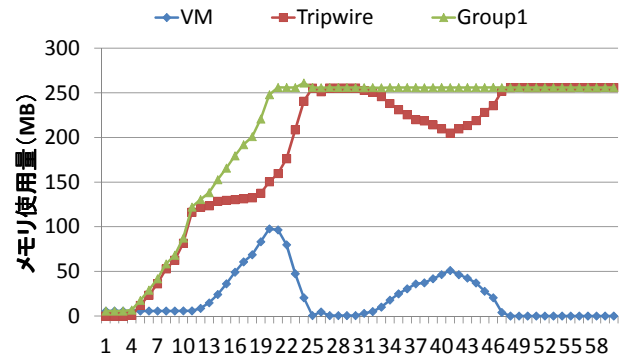


図 14 メモリ使用量の変化

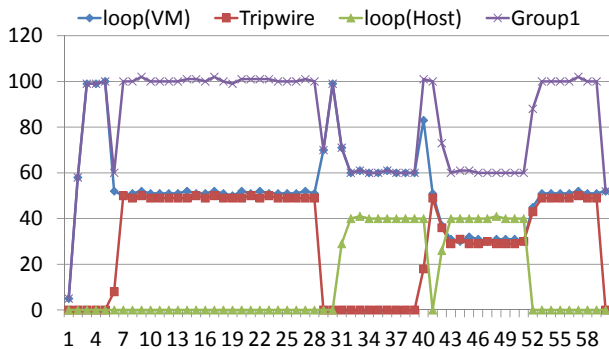


図 13 CPU 使用率の変化 (パターン 2)

合について調べた。図 13 はその時の CPU 使用率の変化を示している。(1) の場合には、Tripwire と loop プログラムが 50 % ずつ CPU を使用した。(2) の場合は、図 12 の場合と同様に 60 : 40 の割合で CPU を使用した。(3) の場合は、グループ間では 60 : 40 の割合で CPU 時間が配分されており、グループ 1 内では 30 % ずつ均等に分け合っていた。

#### 4.3.2 メモリ使用量の制限

VM と IDS のグループ全体のメモリ使用量を制限できることを確かめる実験を行った。VM とオフロードした Tripwire をグループ化し、VM 上でメモリを確保するプログラムを実行した。また、VM と Tripwire のメモリ使用量を個別に把握できるようにするために、VM と Tripwire それぞれからなる子グループを作り、グループをネストさせた。親グループには 256 MB のメモリ制限を設定して実験を行った。図 14 はメモリ使用量の変化を示しており、合計でメモリ使用量を 256 MB 以下に制限出来ていることが分かる。

### 5. 関連研究

Livewire [1] は VM の外から VM 内部の情報を取得する VM イントロスペクションを用いた最初の監視システムである。Livewire ではゲスト OS ごとに OS インタフェース・ライブラリを用意し、ハードウェアの状態から OS の状態を再構築する。Livewire は VMware Workstation に対して実装されており、KVMonitor と同様に IDS をホス

ト OS 上で動作させる。

VMwatcher [3] も VM イントロスペクションを用いた監視システムである。VMwatcher は VM 外での監視結果を VM 内での監視結果と比較する Cross-View Diff を可能にしている。また、ディスクを監視する既存の IDS を VM の外で実行することもできる。VMwatcher は VMware、Xen、QEMU、User-Mode Linux (UML) に対して実装されている。しかし、性能評価は UML についてしか行われておらず、仮想化ソフトウェア間での比較は行われていない。QEMU における実装は KVM でも適用できる可能性があるが、実装の詳細については不明である。

libVMI [15] は KVM と Xen に対応した VM イントロスペクションを行うためのライブラリである。libVMI は Xen 用のライブラリである XenAccess [16] の後継である。libVMI では VM の物理メモリにアクセスするために 2 つの方法を提供している。一つは QEMU にメモリ内容を取得するための QMP コマンドを追加する方法である。もう一つの方法は、メモリダンプを行う QEMU の既存の QMP コマンドを用いる方法である。いずれの方法もネットワーク経由でメモリ内容を送る必要があり、VM のメモリを監視する性能は低いと考えられる。

HyperSpector [2] は OS レベルの仮想化を用いて、IDS オフロードを実現するシステムである。IDS と監視対象システムはそれぞれ、IDS-VM、サーバ VM と呼ばれる仮想環境内で実行される。IDS-VM とサーバ VM は OS を共有しているため、サーバ VM のディスク、ネットワーク、プロセスの監視を容易に行うことができる。

### 6. まとめ

本稿では、KVM における IDS オフロードを実現する KVMonitor を提案した。KVMonitor はホスト OS 上にオフロードした IDS が、VM のメモリやディスク、ネットワークの監視を行うことを可能にする。また、オフロードした IDS が消費するホスト OS のリソースを考慮した VM の性能分離を可能にする。我々は KVMonitor を用いて Transcall を KVM に移植し、既存の IDS をオフロードして異常を検知できることを示した。さらに、KVM と



Xen におけるオフロード性能を定量的に比較した結果、Tripwire の実行を除いて、KVM のほうが性能がよいことが分かった。

今後の課題は、より多くの IDS を動作させられるようにして、さらに網羅的な性能評価を行うことである。

## 参考文献

- [1] Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proc. Network and Distributed Systems Security Symp.*, pp. 191–206 (2003).
- [2] Kourai, K. and Chiba, S.: HyperSpector: Virtual Distributed Monitoring Environments for Secure Intrusion Detection, *Proc. Intl. Conf. Virtual Execution Environments*, pp. 197–207 (2005).
- [3] Jiang, X., Wang, X. and Xu, D.: Stealthy Malware Detection through VMM-based Out-of-the-box Semantic View Reconstruction, *Proc. Conf. Computer and Communications Security*, pp. 128–138 (2007).
- [4] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the Art of Virtualization, *Proc. Symp. Operating Systems Principles*, pp. 164–177 (2003).
- [5] 新井昇鎬, 光来健一, 千葉滋: 仮想マシンを用いた IDS オフロードにおける CPU 資源管理, 第 114 回 OS 研究会 (2010).
- [6] 内田昂志, 岡崎正剛, 光来健一: IDS オフロードを考慮した仮想マシンへの動的メモリ割当, 第 116 回 OS 研究会 (2011).
- [7] 飯田貴大, 光来健一: VM Shadow: 既存 IDS をオフロードするための実行環境, 第 119 回 OS 研究会 (2011).
- [8] 宇都宮寿仁, 光来健一: VM マイグレーションを可能にする IDS オフロード機構, 日本ソフトウェア科学会第 28 回大会 (2011).
- [9] Butt, S., H. L.-C., Srivastava, A. and Ganapathy, V.: Self-Service Cloud Computing, *Proc. Conf. Computer and Communications Security* (2012).
- [10] Murilo, N. and Steding-Jessen, K.: chkrootkit – Locally Checks for Signs of a Rootkit, <http://www.chkrootkit.org/>.
- [11] Kim, G. and Spafford, E.: The Design and Implementation of Tripwire: A File System Integrity Checker, *Proc. Conf. Computer and Communications Security*, pp. 18–29 (1994).
- [12] Roesch, M.: Snort – Lightweight Intrusion Detection for Networks, *Proc. USENIX System Administration Conf.* (1999).
- [13] Norcott, W.: IOzone Filesystem Benchmark, <http://www.iozone.org/>.
- [14] Avallone, S. and Pescapè, A.: D-ITG, Distributed Internet Traffic Generator, <http://traffic.comics.unina.it/software/ITG/>.
- [15] Payne, B. and Leinhos, M.: vmitools – Virtual Machine Introspection Tools, <http://code.google.com/p/vmitools/>.
- [16] Payne, B., Carbone, M. and Lee, W.: Secure and Flexible Monitoring of Virtual Machines, *Proc. Annual Conf. Computer Security Applications*, pp. 385–397 (2007).