KVM における仮想マシンを用いた IDS オフロードの実現

中村　孝介 †　　光来　健一 †‡

† 九州工業大学　　‡ 独立行政法人 科学技術振興機構, CREST

攻撃を検知するために侵入検知システム（IDS）が用いられているが、IDS も攻撃によって無効化される可能性がある。IDS への攻撃を防ぐために近年、IDS と監視対象システムを別々の仮想マシン（VM）で動作させる IDS オフロード手法が提案されている。これまで IDS オフロードの研究は主に Xen を用いて行われてきた。本稿では KVM を用いて IDS オフロードを実現する KVMonitor を提案する。KVMonitor はオフロードした IDS が VM のディスクとメモリの監視を行うことを可能にし、IDS オフロードを考慮した VM の性能分離を実現する。

IDS Offloading with Virtual Machines in KVM

Kosuke Nakamura†　　Kenichi Kourai†‡

†Kyushu Institute of Technology　　‡JST, CREST

Abstract Although instrusion detection systems (IDSes) are used to detect attacks, they may be also attacked. To protect them, IDS offloading has been recently proposed. It runs IDSes and their target system in different virtual machines (VMs). So far, researches on IDS offloading are mainly based on Xen. In this paper, we propose KVMonitor for IDS offloading in KVM. KVMonitor enables IDSes to monitor disks and memory in VMs and achieves performance isolation among VMs, considering IDS offloading.

1　はじめに

インターネットに接続されたホストへの攻撃を検知するために侵入検知システム（IDS）が用いられている。IDS はホスト内のディスクやメモリを監視したりネットワーク通信を監視したりすることにより、攻撃の兆候を検知して管理者に通知する。一方、攻撃者は IDS に検知されるのを防ぐために、ホストへの侵入後に IDS の無効化や改竄を試みることが増えてきた。

このような IDS 自身への攻撃に対処するために、近年、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]。この IDS オフロードは、IDS を監視対象 VM とは別の VM 上で動作させ、オフロード元の VM を監視させる。IDS のオフロードを行うことにより、監視対象 VM に侵入した攻撃者によって IDS が無効化されることを防ぐことができ、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主に Xen を用いて行われてきた [2][5][6]。一方で、最近普及してきた KVM において IDS オフロードはまだほとんど研究されていない。KVM はクラウド等でも急速に使われるようになりつつあるため、IDS オフロードを実現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。しかし、KVM のアーキテクチャは Xen とは大きく異なるため、Xen におけるオフロード手法が適用できるかどうかが不明であった。

本稿では KVM を用いて IDS のオフロードを実現する KVMonitor を提案する。KVMonitor

では監視対象 VM のディスクとメモリの監視を行うことを可能にしている。また、Linux の Cgroups 機能を用いることで、IDS をオフロードしても IDS と VM をひとまとまりとした性能分離を可能にする。

以下、2 章では Xen における IDS オフロードおよび KVM について述べる。3 章では KVMonitor の設計と実装について説明し、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 では通常の VM であるドメイン U から特権を持った VM であるドメイン 0 に IDS をオフロードする。ドメイン 0 はドメイン U の管理を行う権限を持っており、仮想デバイスも提供しているため、ドメイン 0 で動作する IDS はドメイン U を監視することができる。逆に、ドメイン U からドメイン 0 へのアクセスはできないため、ドメイン U が攻撃を受けたとしても IDS を無効化することはできない。

ドメイン 0 にオフロードした IDS は、ドメイン U のディスク、ネットワーク、メモリなどの監視を行うことができる。ディスクの監視はドメイン U のディスクイメージをドメイン 0 にマウントして、マウントしたディレクトリに対して Tripwire などを用いる。ドメイン 0 に作られる仮想ネットワークインターフェースを Snort などを用いて監視することで、ドメイン U が送受信するパケットの監視を行うことができる。メモリについては、ドメイン U のメモリページをドメイン 0 の IDS プロセスのアドレス空間にマップすることで監視することができる。ドメイン U のメモリから OS カーネル内の情報を取得するために、VM イントロスペクション [1] と呼ばれる技術を用いる。

2.2 IDS オフロードを考慮した性能分離

VM を用いて IDS をオフロードすると VM 間での性能分離が正確に行えなくなる [5]。IDS は元々、オフロード元の VM 内で動いていたため、オフロードしても VM の一部として扱えるようにするのが望ましい。しかし、オフロードした IDS はドメイン 0 の資源を使用するため、VM と IDS をひとまとまりとした資源の利用制限が難しくなる。

この問題を解決するために、OffloadCage や Balloon Performer が提案されている。OffloadCage[5] は IDS プロセスとオフロード元の VM を CPU 資源管理の単位としている。この資源管理の単位に対して CPU 使用率の上限等を設定することで、IDS をオフロードする前と同様の性能分離を実現することができる。Balloon Performer[6] はオフロードした IDS が使用するメモリ量を測定してオフロード元 VM のメモリを動的に割り当て直す。

2.3 KVM

KVM は Linux カーネルの中で仮想マシンモニタを動作させ、VM をホスト OS の 1 つのプロセスとして管理する。そのため、仮想マシンモニタは Linux カーネルに実装されている機能を使用することができる。KVM は CPU の仮想化支援を利用する完全仮想化のみに対応している。VM にディスク、ネットワーク、メモリを提供するために QEMU というエミュレータを利用している。

一方、Xen では仮想マシンモニタは独立したソフトウェアであり、その上では VM のみが動作する。完全仮想化にも対応しているが、ゲスト OS に修正を加えて動作させる準仮想化が用いられることが多い。準仮想化の場合には、VM のディスクやネットワークはドメイン 0 が提供する。VM のメモリは仮想マシンモニタが提供する。Xen の仮想マシンモニタは最小限の機能のみを提供している。

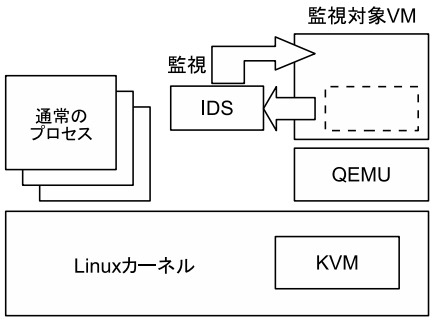

図 1: KVM のアーキテクチャ

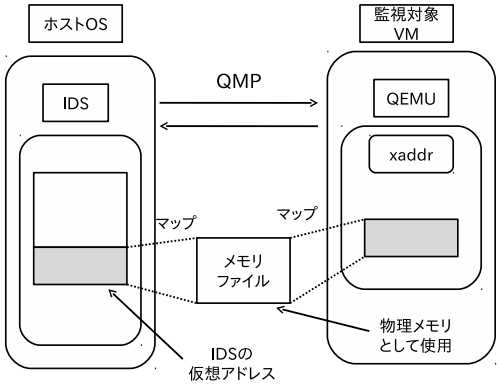

図 2: メモリファイルのマップ

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 IDS をホスト OS 上にオフロードし、VM の監視を行うことを可能にする。現在のところ、監視対象 VM のディスクとメモリの監視を行うことができる。また、KVM において VM は 1 プロセスであることを利用して、Linux カーネルの機能を用いて IDS オフロードを考慮した性能分離を行う。

3.1 ディスクの監視

オフロードした IDS が監視対象 VM のディスクを監視できるようにするために、KVMonitor は VM のディスクをホスト OS 上にマウントする。KVM では qcow2 と呼ばれる形式のディスクイメージを使用することが多い。しかし、qcow2 形式のディスクイメージは直接マウントすることができないため、ネットワークブロックデバイス（NBD）を利用する。NBD は別のホストのブロックデバイスをネットワーク経由でローカルのブロックデバイスとして扱うことを可能にし、qcow2 形式にも対応している。

現在の実装では、qemu-nbd という NBD サーバと nbd-client という NBD クライアントを用いている。qemu-nbd は nbd-client からの要求に応じて、qcow2 形式のディスクイメージをマウントできる形式である raw 形式に変換し、そのディクスブロックを返す。nbd-client は qemu-nbd から受け取ったディスクブロックをブロックデバイスとして提供する。さらに、kpartx コマンドを利用してパーティション毎のブロックデバイスを作成し、vgchange コマンドで論理ボリュームの活性化を行う。

Xen におけるドメイン U のディスクイメージは raw 形式で作成されるのが一般的であるため、NBD を用いずにそのままマウントすることができる。ディスクイメージを NBD としてマウントする手法は直接マウントする手法と比較すると性能が低下すると考えられる。

3.2 メモリの監視

オフロードした IDS が VM 内部のメモリを監視できるようにするために、KVMonitor は VM の物理メモリをファイルとして IDS に提供する。IDS はこのファイルを mmap システムコールを用いてメモリマップすることで、VM のメモリにアクセスする。KVM では VM の一部である QEMU が VM のメモリを確保しており、VM 以外のプロセスが VM のメモリにアクセスすることはできない。そこで、QEMU にメモリ用のファイルを作成させ、このファイルをマップしてメモリとして使わせる。ただし、従来の QEMU はこのファイルをオープンした後で削除していたため、IDS からアクセスできるように QEMU に修正を加えた。

仮想アドレスを使って VM のメモリ上のデータにアクセスするために、IDS は QEMU と通信して仮想アドレスを物理アドレスに変換する。VM 内部の OS カーネルの変数の仮想アドレス

は分かっているが、VM のメモリには物理アドレスでアクセスする必要がある。QEMU はゲスト OS のページテーブルを調べて仮想アドレスを物理アドレスに変換する関数を持っているため、QEMU 外部から変換を行えるようにする xaddr コマンドを追加した。IDS は得られた物理アドレスをオフセットとして、メモリ用ファイルにアクセスする。

IDS から QEMU の xaddr コマンドを呼び出すには、QEMU Monitor Protocol（QMP）を使って QEMU と通信を行う。QMP は JSON を用いてアプリケーションと QEMU の間で通信を行うプロトコルである。IDS は QEMU の QMP 用ポートに接続した後、xaddr コマンドの実行を要求する。QEMU はこのコマンドを実行して、変換した物理アドレスを返す。

Xen ではドメイン 0 の IDS がドメイン U のページテーブルを参照して仮想アドレスを物理アドレスに変換する。この物理アドレスを含むメモリページをドメイン 0 の IDS のアドレス空間にマップすることで、IDS からドメイン U の指定したメモリアドレスを参照することができる。このように Xen では VM のメモリを直接マップすることができるが、ページテーブルのメモリページもアクセスする度にマップする必要がある。

KVM ではメモリ用のファイルを用いることによるオーバーヘッドを減らすために、このファイルを hugetlbfs というファイルシステム上に置く。hutetlbfs は Linux のヒュージページ機構を利用しており、メモリマップの際のページサイズを 4KB から 2MB にする。これにより、ページテーブル探索のコストを減らし、TLB への影響を抑えることができる。

3.3 性能分離

IDS とオフロード元 VM をひとまとまりとして性能分離を行えるようにするために、KVMonitor は Linux カーネルが提供している 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の資源管理を可能にする。Cgroups を用いて、ホスト OS 上で IDS と監視対象 VM の 2 つのプロセスをグループ化する。KVM では VM が 1 プロセスとして作成されるため、このようなグループ化を容易に行うことができる。

このグループに対して CPU やメモリに対する制約を設定することで性能分離を行う。VM と IDS のグループに対して CPU 使用率を設定するには、cpu.shares に割り当てる CPU の割合を指定する。他のグループに設定した CPU の割合に依存するが、VM と IDS のグループに割り当てる CPU の下限を相対的に設定することができる。例えば、VM と IDS のグループとその他のグループに CPU を 1：1 で割り当てた場合、それぞれのグループは CPU を 50 %以上使えることが保証される。

VM と IDS が使うメモリの上限を設定するには、memory.limit_in_byte に上限値を指定する。Cgroups はプロセスが使っているメモリだけでなく、プロセスのファイルアクセスによってカーネル内に作られたファイルキャッシュも考慮する。そのため、ファイルキャッシュのサイズも含めて、設定した上限を超えないように制限される。ただし、QEMU プロセス自身が使っているメモリや確保したファイルキャッシュも考慮する必要がある。

Xen では、IDS オフロード時の性能分離を実現するために、IDS の使用する資源量を測定し、VM の資源管理に反映させる必要がある。CPU に関する性能分離を行う OffloadCage では仮想マシンモニタ内の VM スケジューラを変更している。メモリに関する性能分離を行う Balloon Performer では、IDS によって確保されたファイルキャッシュサイズを測定するためにドメイン 0 の OS カーネルを修正している。OffloadCage では Credit Scheduler の機能を用いて、VM と IDS のグループに対して CPU 使用率の上限を設定することもできる。

4 実験

オフロードした IDS から VM のディスクとメモリの監視および性能分離に関する実験を行った。実験には CPU に Intel Xeon 2.53GHz の CPU、6GB のメモリを搭載したマシンを使用

表 1: Tripwire の実行時間

	実行時間（分）
オフロードなし	9.2
オフロードあり	9.9

した。VM には 30GB のディスクと、512MB のメモリを割り当てた。

4.1 Tripwire を用いた監視

Tripwire を用いて監視対象 VM のディスクの整合性を検査する実験を行った。Tripwire をホスト OS 上にオフロードした場合でも、VM のディスクの検査を行うことができ、追加・削除・変更されたファイルを検出できることが確認できた。

次に、Tripwire をオフロードした場合としなかった場合とで、ディスクの検査にかかる時間を測定した。表 1 にそれぞれの実行時間を示す。オフロードした方が実行時間が若干長くなったが、これは VM のディスクイメージを NBD 経由でマウントしているため、ファイルアクセスに時間がかかることが原因であると考えられる。

4.2 システムコールテーブルの監視

ホスト OS から VM のカーネルメモリ上にあるシステムコールテーブルの監視する時間の測定した。システムコールテーブルを監視するには、その仮想アドレスを物理アドレスに変換し、VM のメモリを読む必要がある。システムコールテーブルを読み出す IDS の実行を 1,000 回繰り返したところ、1 回の実行にかかる時間は 1.5 ミリ秒であった。アドレス変換にかかる時間は 1.3 ミリ秒であったことから、ほとんどの時間はアドレス変換に費やされていることが分かった。

4.3 CPU に関する性能分離

オフロードした IDS と監視対象 VM のグループに対して CPU 使用率の下限が設定できることを確かめる実験を行った。IDS として Tripwire を使用し、VM 上とホスト OS 上で無限ループを行うプログラムを実行した。Tripwire と VM のプロセスを Group1、ホスト OS 上で無限ループするプロセスを Group2 とし、Group1 と Group2 に割り当てる CPU の割合を 40:60 とした。実験結果をわかりやすくするために使用する CPU を 1 つに制限した。

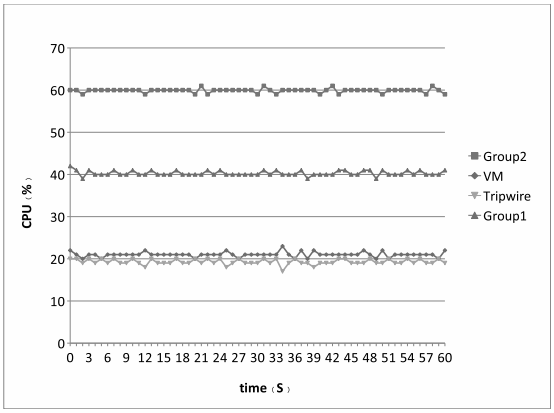

図 3: VM と Tripwire の CPU 使用率

図 3 に Group1 と Group2 の CPU 使用率と Group1 中の Tripwire と VM のそれぞれの CPU 使用率を示す。Group1 の CPU 使用率は 40%でほぼ一定していることがわかる。

4.4 メモリに関する性能分離

オフロードした IDS と VM のグループに対して使用するメモリの上限が設定できることを確かめる実験を行った。IDS として Tripwire を使用し、Tripwire と VM のプロセスを Group1 とした。さらに、物理メモリの上限を 512MB に設定した。図 4 に Group1 のメモリ使用量の変化を示す。Group1 のメモリ使用量の最大値は 512MB を超えないように制御できていることがわかる。

5 関連研究

Livewire[1] では VMware を用いて IDS をオフロードし、ディスク、ネットワーク、メモリの監視を行う。VM のメモリにアクセスするた

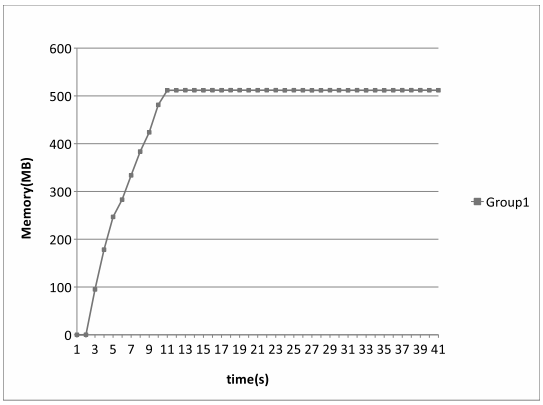

図 4: Tripwire のメモリ使用量

めに DMA を用いている。VMwatcher[2] では VMware、Xen、QEMU、UML を用いて IDS をオフロードし、ディスクとメモリの監視を行う。QEMU を用いた実装は KVM にも適用可能である。KVMonitor と同様に QEMU を拡張して、ファイルを VM のメモリとして使わせている。一方、アドレス変換はゲスト OS のページテーブルをたどることで行っている。KVMonitor では QEMU 自身にアドレス変換を行わせることで、アーキテクチャ依存度を減らしている。

HyperSpector[4] は OS レベルの仮想化を用いて IDS をオフロードし、ディスク、ネットワーク、システムコールの監視を行う。システムレベルの仮想化と違い、IDS がオフロード元の仮想環境と同じ OS を共有するため、これらの資源の監視は比較的容易である。

Flicker や SPE Observer では AMD-V や Cell/B.E. などのハードウェアを用いて IDS おオフロードし、メモリの監視を行う。Flicker[3] ではトラステッド・コンピューティング技術を用いて IDS を隔離された環境で実行する。SPE Observer[7] では IDS を SPE 上で動作させ、Isolation モードを用いて隔離する。

6 まとめ

本稿では KVM における IDS オフロードを実現するシステム KVMonitor を提案した。KVMonitor では、NBD を用いることで IDS によるディスクの監視を可能にし、QEMU を改造することで IDS によるメモリの監視を可能にする。さらに、Linux の Cgroups 機能を用いることで、IDS オフロード時の VM の性能分離を実現する。

今後の課題は、ネットワーク IDS のオフロードに対応することである。また、Xen で実現されている IDS オフロードとの定量的な性能比較を行う予定である。

参考文献

[1] T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection Based Architecture for Instrusion Detection. *Proc. Network and Distributed Systems Security Symp.*, pp.191-206, 2003.

[2] X. Jiang, X. Wang, and D. Xu. Stealthy malware detection through vmm-based ”out-of-the-box” semantic view reconstruction. In *Proc. Conf. Computer and Communications Security*, pp.128-138, 2007.

[3] J. M. McCune, B. Parno, A. Perring, M. K. Reiter, and H. Isozaki. Flicker: An Execution Infrastructure for TCB Minimization. In *Proc. European Conf. Computer Systems*, pp.119-128, 2008.

[4] K. Kourai and S. Chiba: HyperSpector: Virtual Distributed Monitoring Environments for Secure Instrusion Detection, In *Proc. Int. Conf. Virtual Execution Environments*, pp.197-207, 2005.

[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 年.

[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 年.

[7] 永田卓也, 光来健一, Cell/B.E. の SPE 上で動作する安全な OS 監視システム, 日本ソフトウェア科学会第 27 回大会, 2010 年.