Vol.2013-DPS-154 No.30
Vol.2013-CSEC-60 No.30
2013/3/14

管理VM監視のためのメモリアクセス監視機構の開発

猪飼 淳[1]　齋藤 彰一[1]　毛利 公一[2]　松尾 啓志[1]

概要：IaaS 型クラウドにおいてユーザが利用する計算資源は仮想マシン (VM) であり，VM はサービス提供者の管理 VM によって管理される．クラウド管理者は管理 VM の権限を行使することでユーザ VM 内のメモリ内の情報を覗き見ることが可能であるが，一方でユーザはメモリが管理 VM に覗かれたことを検知する術を持たない．本稿では仮想マシンモニタ (VMM) が管理 VM のユーザ VM へのメモリの覗き見を検知し，ユーザ VM へ通知するシステムを提案する．ユーザ VM は管理 VM による覗き見を禁止するメモリ領域を指定することができ，ユーザ VM のメモリすべてを暗号化する場合と比べて低負荷でクラウドの運用が可能になる．

1. はじめに

クラウドコンピューティングの普及により，ネットワークを経由して計算資源を利用するケースが増えている．中でも Infrastructurte as a Service (IaaS) 型クラウドでは，ユーザはネットワークを経由してサービス提供者が提供する仮想マシン (VM) を利用する．これにより，ユーザは計算資源を保有する必要がなくなり，計算資源のメンテナンス等といったメリットを享受することができる．

ユーザが利用する仮想マシン (ユーザ VM) はクラウドサービス提供者が保有する仮想マシン (管理 VM) によって管理される．管理 VM は管理者権限を用いて VM の起動，終了，サスペンド，マイグレーションと言った操作を行い，ユーザ VM の管理を行う．

管理 VM に与えられる特権は非常に強く，ユーザ VM メモリに自由にアクセスすることが可能である．そのため，ユーザにとってはユーザ VM 内のメモリ内容が管理 VM に漏洩するリスクが存在する．また，このような，管理 VM によって実行されるユーザ VM メモリへのアクセスは，ユーザ VM からは隠蔽されており，管理 VM によるメモリアクセスがあったことをユーザ VM は検知することはできない．

この問題を解決するため，管理 VM によるユーザ VM メモリへのアクセスを仮想マシンモニタ (VMM) が検知し，ユーザ VM に通知するシステムを提案する．さらに，管理 VM によるユーザ VM メモリの検査の可否を予め指定するために，ユーザ VM は VMM に対して，管理 VM によるメモリアクセスを拒否するメモリ領域を指定可能とする．ユーザ VM によって指定された領域に対して管理 VM がメモリアクセスを試みた場合には，メモリアクセスを失敗させたり，メモリを暗号化することによってユーザ VM のメモリを保護する．この指定により，VMM によるメモリアクセス制御は，すべての領域を暗号化で保護する場合と比べて VMM の負荷が小さくなる．これにより，サスペンド時間の短縮や，仮想化環境全体でのパフォーマンス向上が可能である．

本提案システムを Xen 4.1.2 上に実装した．管理 VM によるユーザ VM メモリへのアクセスを検知し，ユーザ VM へ管理 VM によるメモリアクセスの履歴を通知する．さらに，ユーザ VM による指定領域のメモリアクセスの拒否が実現できていることを確認した．

以下，2 章で関連研究について述べる．3 章では提案システムについて述べ，4 章ではシステムの詳細な実装について述べる．5 章では提案システムの動作を確認するために行った実験について述べる．6 章ではまとめと今後の課題について述べる．

2. 関連研究

2.1 管理 VM によるユーザ VM メモリへのアクセス

IaaS 型クラウドにおいてユーザはネットワークを経由して VM にアクセスする．ユーザがアクセスする計算資源はクラウドサービスを提供するクラウド管理者によって管理されている．クラウド管理者は管理 VM と呼ばれる特権を与えられたクラウド管理者専用の VM を用いて管理を行う．管理 VM は特権を用いて，VM の起動，終了，サスペンド，マイグレーションといった操作によりユーザ VM の管理を行う．これらの操作には，管理 VM はユーザ VM の

1　名古屋工業大学 466-8555 愛知県名古屋市昭和区御器所町
2　立命館大学 525-8577 滋賀県草津市野路東 1 丁目 1-1

ⓒ 2013 Information Processing Society of Japan

メモリへアクセスする必要があるため，管理 VM はユーザ VM 内のすべてのメモリ領域にアクセスする権限が与えられている．

管理 VM のメモリアクセス権限はユーザ VM の管理以外にも用いることができる．Virtual Machine Introspection(VMI)[1] はメモリアクセス権限を利用した技術の一つであり，ユーザ VM 内の OS レベルのデータ構造を VMM や管理 VM 上から識別する技術である．VMI を用いた例として，管理 VM からユーザ VM では検知できない rootkit を検出するシステム [2] や，ユーザ VM 上で動作している侵入検知システムをオフロードし管理 VM 上で動作させるシステム [3] が提案されている．また，悪意を持ったクラウド管理者は，特権 VM のメモリアクセス権限を用いることでユーザ VM メモリ上の機密情報を自由に盗むことが可能であり，これを防止する手法 [4][5] が提案されている．

2.2 ユーザ VM メモリの保護 VMCrypt

VMCrypt[4] はユーザ VM のメモリを暗号化することにより，管理 VM へのメモリ内の機密情報の漏洩を防止する手法である．VMCrypt は，管理 VM がユーザ VM のメモリへアクセスしようとした際に VMM がユーザ VM のメモリを暗号化する．一方，ユーザ VM が自身のメモリへアクセスを試みた際は暗号化を行わない．

VMCrypt の問題点として以下の 3 点が挙げられる．第 1 に，ユーザ VM 自身がメモリが覗かれたことを知ることができないという点である．VMCrypt では，メモリ覗き見があったことを検知するのは VMM でありユーザ VM は一切関与しない．そのため，ユーザ VM は情報漏洩の可能性のある環境にいることを認識することができない．

第 2 に，管理 VM によるメモリアクセスは必ずしも不正行為にのみ用いられるわけではなく，VMI のようにセキュリティ向上のためにアクセスする場合が存在する．VMCrypt ではサスペンドやマイグレーションといった IaaS 運営上に必要なメモリ領域に関しては，非暗号ページとして VM 起動時に登録して暗号化の対象外とすることで，これらの操作を実行可能にしている．しかし，VMI を用いたシステムに関しては対象外であるため，メモリを暗号化することによって動作不可能になる．

最後に，暗号化による VMM への負荷も無視することができない．メモリの暗号化を行った場合，通常のメモリアクセスと比べて暗号化の計算時間が追加されるため，サスペンドやマイグレーションといった IaaS 運営上に必要な操作に必要な時間は増加する．また，暗号化は VMM が行うため，仮想化環境全体で性能が低下することも問題である．暗号化などの VMM への負荷によって仮想化環境全体で性能が低下することを確認するために予備評価の実験を行った．実験は VMM がサスペンドや暗号化を行っている最中に，サスペンド対象ではない別の VM で文字列出力を繰り返すプログラムが 1 秒間に実行できる回数を測定した．一回の文字列出力では 8 バイト出力し，メモリの暗号化は AES を用いた．測定結果を**図 1** に示す．サスペンド中は通常時と比べて 29%，メモリ暗号化ありのサスペンド中は通常時と比べて 44%低下した．この実験により，VMM による暗号化が仮想化環境全体の性能を低下させることが示された．

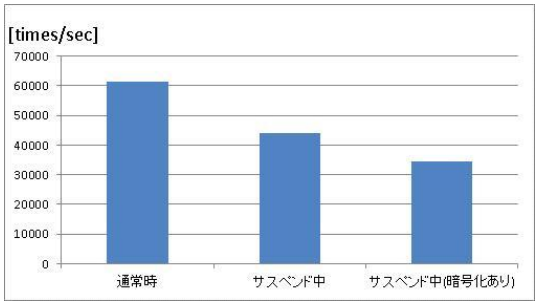

図 1 文字列出力プログラムの性能予備評価

3. 提案手法

ユーザ VM 利用者による管理 VM の監視のために，管理 VM によるメモリアクセスの監視を行い，そのメモリアクセス履歴をユーザ VM へ通知するシステムを提案する．さらに，ユーザ VM がアクセス禁止領域を指定できるシステムを提案することで，暗号化負荷を軽減する．なお，本提案手法は VMM である Xen 上に実装されることを想定している．

3.1 メモリアクセスの検知

管理 VM がユーザ VM 内のメモリへアクセスを行う場合，Xen では特権命令であるメモリマップ用ハイパーコールを用いる．ハイパーコールは必ず一度 VMM を経由して実行されるため，VMM 内でメモリマップ用ハイパーコールの実行を監視することで，管理 VM のユーザ VM に対するメモリアクセスを検知することができる．

3.2 メモリアクセス履歴の通知

VMM が管理 VM によるユーザ VM へのメモリアクセスを検知した後，管理 VM が行ったメモリアクセスの履歴をユーザ VM に通知する．これは VMM と一般 VM との共有メモリを作成することによって実現する．VMM はメモリアクセスを検知した際に，アクセス履歴を共有メモリに書き込む．ユーザ VM は，共有メモリからアクセスの履歴を読み取ることによって管理 VM によるメモリアクセスを知ることができる．また，アクセス履歴のための共有メモリが改ざんされることを防止するために，作成する共有メモリは管理 VM によってアクセスされないようにする必要がある．この仕組みは次節で述べるメモリアクセス制御リストによって実現する．

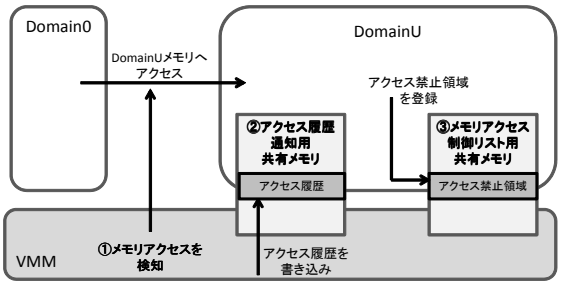

図 2 システムの全体構成

3.3 メモリアクセス制御リスト

VMM は, 管理 VM によるメモリアクセスを検知した際にメモリマップ用ハイパーコールを失敗, もしくは VMCrypt と同様に暗号化することでユーザ VM メモリの内容が管理 VM に漏洩することを防止する．しかし，ユーザ VM へのメモリアクセスすべてを禁止した場合，サスペンドやマイグレーションといった IaaS 運営上必要な管理操作や，VMI を利用したセキュリティシステムの管理 VM へのオフロード手法を正常に動作させることが困難になる．そのため，ユーザ VM は，VMM に対して管理 VM によるメモリマップを拒否したい領域を指定し，ユーザ VM によるアクセス制御を実現する．この機能はメモリアクセス履歴の通知と同様に VMM とユーザ VM 間の共有メモリを用いる．以降，この共有メモリをメモリアクセス制御リストという．VMM は管理 VM によるメモリアクセスを検知した際，メモリアクセス制御リストを探索し，メモリアクセスが禁止されているページとアクセス履歴通知用の共有メモリ領域の場合はメモリアクセスを失敗させる．これにより，ユーザ VM は予め管理 VM によって盗み見られたくない領域を VMM に指定することが可能になり，領域を指定するセキュリティポリシ次第で，VMI を用いたセキュリティシステムを動作させることも可能になる．

また，メモリアクセス制御リストにより，アクセス制御対象外の領域に対しては暗号化を行わないため，VMM が暗号化を行う頻度が低下することによって，サスペンド速度の向上や仮想化環境全体での性能向上が期待できる．

4. 実装

提案システムを Xen 4.1.2 上に実装した．Xen では仮想マシンの単位をドメインと呼び, 特に管理 VM を Domain0, ユーザ VM を DomainU と呼ぶ．また，ユーザ VM 上で動作するゲスト OS は準仮想化 Linux を対象とする．

4.1 全体構成

提案システムは，大きく分けて**図 2** のように 3 つの機構によって構成される．1 つ目はメモリアクセスを検知する機構で，Domain0 が DomainU のメモリへアクセスした際に VMM が検知する部分である．2 つ目はアクセス履歴通知用共有メモリで，VMM-DomainU 間の共有メモリを作成し，共有メモリを介してアクセス履歴の通知を行う．3 つ目はアクセス制御リスト用共有メモリで，アクセス履歴通知用共有メモリと同様に VMM-DomainU 間の共有メモリを作成し，DomainU は共有メモリにアクセス禁止領域を登録し，VMM は登録された禁止領域をもとにアクセス制御を行う．

4.2 メモリアクセスの検知

Domain0 が DomainU のメモリにアクセスする際，必ずメモリマップ用ハイパーコールが実行される．ハイパーコール呼び出しによって，VMM へ処理が移り，VMM はページテーブルエントリの変更を行う．そのため VMM 内でのページテーブルエントリ操作を監視することで Domain0 による DomainU メモリへのアクセスを VMM は検知することができる．

ページテーブルエントリの変更を検知するために Xen のセキュリティフレームワークである Xen Security Modules(XSM)[6] を用いた．XSM は VMM 内で発生する主要なイベントをフックすることで, 強制アクセス制御を行うためのフレームワークである．ここで, メモリマップによるページテーブルエントリ更新の際は xsm_mmu_nornal_update 関数を経由する．ページテーブルエントリ更新操作は Domain0 による DomainU へのメモリマップ以外の場面でも発生するが，xsm_mmu_nornal_update 関数に渡される引数には，メモリマップの対象となっているドメインとページの所有者とページテーブルエントリが指定されており，これらを調べることで Domain0 による DomainU へのメモリマップのみを特定することができる．

4.3 VMM-DomainU 間の共有メモリの作成

VMM が Domain0 による DomainU メモリへのアクセスの検知した後，アクセス履歴を DomainU に伝えるために VMM-DomainU 間の共有メモリを用いる．共有メモリとして用いるメモリ領域は DomainU が作成し，そのアドレスを VMM に伝えることによって VMM からアクセス可能にする．本システムでは，独自なハイパーコールを新た追加せずに，XSM と Xen のドメイン間メモリ共有機構である Grant Table を用いて実装する．

Grant Table とはドメイン間でのページ共有やページ移送を行うための機構である．例えば，DomainU のフロントエンドドライバと，Domain0 のバックエンドドライバがデータをやりとりする際に利用されている．Grant Table を用いて自身のメモリ領域の一部を共有メモリとするには，gnttab_foreign_access 関数を実行する．この関数で，メモリを共有するドメインの ID とページを Grant Table に登

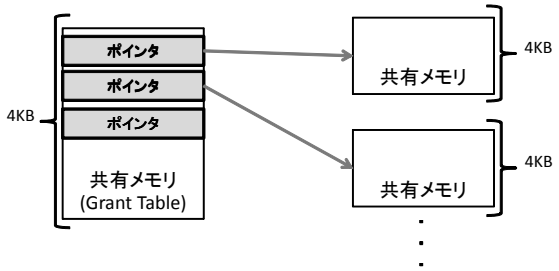

図 3 共有メモリの構造

録する．登録後, 共有を許可されたドメインが Grant Table マップ用ハイパーコールを発行することで，先に Grant Table で指定されたメモリ領域がマップされる．これにより，ドメイン間の共有メモリが実現されている．

このように Grant Table はドメイン間通信に用いる機構であり，本来は VMM と通信を行うために用いられない．しかし, Grant Table 自体には VMM からアクセスすることが可能であり，VM が Grant Table をマップするハイパーコールは XSM によって容易にフックすることが可能である．本システムでは, DomainU が gnttab_foreign_access 関数によって VMM-DomainU 間共有メモリを Grant Table に登録した後に，DomainU は先程自身が登録した Grant Table に対して Grant Table マップ用ハイパーコールを発行する．VMM はこの自己呼び出しの Grant Table マップ用ハイパーコールを XSM によってフックし，DomainU が登録した Grant Table を調べることで共有メモリのアドレスを特定する．

また，Grant Table はマシンフレーム単位でメモリを共有する．本システムを実装した x86_64 版の Xen においてマシンフレームサイズは 4KB であり，後述するアクセス履歴の通知や，メモリアクセス制御リストとして用いるためにはサイズが小さい．そのため，本システムでは**図 3** のように Grant Table として VMM に渡されるマシンフレームには DomainU 上の別のマシンフレームを指すポインタを格納することで，大容量の共有メモリを利用可能にした．

4.4 メモリアクセス履歴の通知

VMM は DomainU から伝えられた VMM-DomainU 間共有メモリの領域を特定した後に, Domain0 から DomainU メモリへのアクセスを検知した際，アクセスカウント，アクセスされたマシンフレーム番号，アクセスされたページの種類をメモリアクセス履歴として共有メモリに書き込む．

DomainU は VMM-DomainU 間共有メモリを VMM に通知後に，アクセス履歴読み込み用のカーネルスレッドを起動させる．カーネルスレッドはビジーウェイトで VMM-DomainU 間共有メモリを調べ，VMM が書き込むメモリアクセス履歴を入手する．VMM-DomainU 間共有メモリはリングバッファで実装されており，DomainU は書きこまれた総アクセス数を確認することによって読み込み漏れがないか確認する．

4.5 メモリアクセス制御リスト

メモリアクセス制御リストはメモリアクセス履歴通知と同様に VMM-DomainU 間の共有メモリによって実現する．なお，メモリアクセス制御リストに用いる領域はメモリアクセス履歴通知用とは別の領域に作成する．DomainU はメモリアクセス制御リストに，Domain0 からアクセスを拒否したいマシンフレーム番号を書き込む．一方，VMM は，Domain0 から DomainU メモリへのアクセスを検知した際にメモリアクセス制御リストを探索し，Domain0 がアクセスしようとしているマシンフレーム番号とメモリアクセス制御リストを比較する．もし，マシンフレーム番号が一致した場合，ページを暗号化もしくは，メモリマップを失敗させることで DomainU メモリ内の情報が Domain0 に渡ることを防止する．

4.6 非暗号化ページ

ユーザ VM が誤ってメモリアクセス制御リストに IaaS 運用に必要なメモリ領域を禁止領域に指定する可能性がある．なお IaaS 運用に必要な領域とは管理 VM によるユーザ VM のサスペンドやマイグレーション，ライブマイグレーションといった操作を指す．本システムでは VMCrypt と同様に VMM はあらかじめ，ユーザ VM 起動時および動作中に IaaS 運用に必要なメモリ領域をビットマップを用いて管理する．

Domain0 から DomainU メモリへのアクセスを VMM が検知した際にメモリアクセス制御リストを探索し, Domain0 がアクセスしようとしているマシンフレーム番号とメモリアクセス制御リストを比較する．一致した場合，VMM はさらにそのマシンフレーム番号と非暗号化ページ登録用のビットマップを比較し，暗号化の可否を判断する．非暗号化ページとして登録されていた場合は，IaaS 運用に必要なメモリ領域であると判断する．その結果，ユーザ VM が禁止領域として指定したとしても管理 VM によるメモリアクセスを許可する．一方，暗号化ページとして登録されていた場合は IaaS 運用に必要なメモリ領域ではないので，管理 VM によるメモリアクセスは許可しない．なお，非暗号化ページとして指定すべきページおよび，ビットマップへの登録方法は VMCrypt[4] と同様の方法を用いた．

4.7 メモリアクセス制御リストのユーザインタフェース

VMM と DomainU の Linux では，それぞれ取り扱うアドレスが異なる．VMM はマシンフレーム番号を用いるが，DomainU の Linux は仮想アドレスを用いる．このため，DomainU の Linux が，メモリアクセス制御リストにア

図 4 デバイスファイル形式のユーザインタフェース

クセス禁止メモリを登録する際に，仮想アドレスからマシンフレーム番号に変換する必要がある．準仮想化の Linux ではカーネルコードの中に仮想アドレスからマシンフレーム番号への変換関数があるため，カーネル空間では容易に変換することができるが，ユーザ空間で動作するユーザプログラムからはこの変換関数を用いることができない．また，メモリアクセス制御リストはカーネルメモリ上に作成されるため，ユーザプログラムからは通常はアクセスすることができない．

そこで，本システムでは**図 4** に示すようにデバイスファイル形式によるアクセス方法を採用した．デバイスファイル形式により，ユーザプログラムはデバイスファイルに対して，read システムコールや，write システムコールを発行することでメモリアクセス制御リストにアクセスすることができる．また，仮想アドレスからマシンフレーム番号へのアドレス変換はカーネル空間で動作するデバイスに任せることができるため，ユーザプログラムは Domain0 からのアクセスを拒否したい領域をアドレス変換を意識せずに指定することができる．

本システムではアクセス拒否領域の指定方式としてアドレス指定方式と，プロセス ID 指定方式を実装した．アドレス指定方式では，ユーザはメモリアクセスを拒否したい仮想アドレスをデバイスファイルに書き込む．その際，デバイスは仮想アドレスをマシンフレーム番号に変換し，メモリアクセス制御リストに書き込む．アドレス指定方式を用いることで，例えば，ユーザプログラムでパスワードを保管するメモリ領域を確保した際に，パスワードを保管する領域の仮想アドレスをデバイスに渡すことで，その領域を Domain0 から覗かれることを防止するという利用方法が考えられる．次に，プロセス ID 指定方式ではユーザはメモリアクセスを拒否したいプロセスのプロセス ID をデバイスファイルに書き込む．その際，デバイスはプロセス ID から該当プロセスのプロセスの構造体である task_struct 構造体が含まれるマシンフレーム番号を特定し，メモリアクセス制御リストに書き込む．プロセス ID 指定方式により，機密情報を扱うプロセスを登録することで，Domain0 が VMI によりそのプロセスの状態等を探ることを困難にすることができる．しかし，task_struct 構造体には，プロセスのメモリを管理している構造体やオープン中のファイルを管理している構造体へのポインタなど多くのポインタが含まれているために，task_struct 構造体をアクセス禁止にするだけではプロセスのデータすべてを保護できない．しかし，task_struct 構造体へのアクセスを禁止することによって，これらのポインタを辿ることが困難になるため，十分と判断し，ポインタの先の領域に対してはアクセス禁止していない．

表 1 実験環境

VMM	Xen4.1.2
ハードウェア	Core-i3 2.4GHz 4core / 8GB
Domain0	Linux 3.4.4 / VCPU 4core / 6GB
DomainU	Linux 3.4.4 / VCPU 1core / 1GB / 準仮想化

```
PID:[     1] COMM:init     PATH:/sbin/init
PID:[     2] COMM:migration/0
PID:[     3] COMM:ksoftirqd/0
PID:[     4] COMM: watchdog/0
```

図 5 アクセス制御未導入時の管理 VM の画面出力の一部分

```
[count:226 mfn:0x1d01cf page_info:PGT_l4]
[count:227 mfn:0x1fabb4 page_info:PGT_l3]
[count:228 mfn:0x1fabb0 page_info:PGT_l2]
[count:229 mfn:0x1f89be page_info:PGT_l1]
[count:230 mfn:0x1fabae page_info:Other ]
[count:231 mfn:0x1d01cf page_info:PGT_l4]
[count:232 mfn:0x1fabb5 page_info:PGT_l3]
[count:233 mfn:0x1fabb1 page_info:PGT_l2]
[count:234 mfn:0x1fa447 page_info:PGT_l1]
[count:235 mfn:0x1c5d61 page_info:Other ]
[count:236 mfn:0x1d01cf page_info:PGT_l4]
[count:237 mfn:0x1fabb5 page_info:PGT_l3]
[count:238 mfn:0x1fabb1 page_info:PGT_l2]
[count:239 mfn:0x1fa447 page_info:PGT_l1]
[count:240 mfn:0x1c5d60 page_info:Other ]
[count:241 mfn:0x1d01cf page_info:PGT_l4]
[count:242 mfn:0x1fabb5 page_info:PGT_l3]
[count:243 mfn:0x1fabb1 page_info:PGT_l2]
[count:244 mfn:0x1fa447 page_info:PGT_l1]
[count:245 mfn:0x1c5d5e page_info:Other ]
```

図 6 プロセス一覧取得時のユーザ VM ログの一部分

5. 実験

提案システムの動作確認と，暗号化頻度削減による効果の確認のための評価を行った．**表 1** に実験環境を示す．

5.1 メモリアクセス履歴通知機構の動作確認

メモリアクセス履歴通知機構の動作確認を行うために，提案システム導入後にユーザ VM 内のプロセス一覧を取得するプログラムを管理 VM 上で実行した．なお，通知機構

の動作確認であるため，メモリアクセス制御リストには指定していない．管理 VM 上でユーザ VM 内のプロセス一覧を取得するプログラムを実行した際の，管理 VM の出力結果の一部を**図 5**，ユーザ VM 上に表示されたログの一部分を**図 6** に示す．

図 5 のプロセス一覧取得プログラムの実行結果について述べる．PID はプロセス ID を表し，COMM は実行コマンド名，PATH は実行プログラムのパスを表す．なお，PATH はカーネルスレッドの場合は表示されない．

図 6 の 1 行目を例にログの中身の仕様について述べる．1 行目の count は管理 VM によってアクセスされたページの総数を表す．1 行目の場合は，これまで 226 回の管理 VM によるユーザ VM メモリへのアクセスがあったことを表している．mfn は管理 VM によってアクセスされたマシンフレーム番号を表す．1 行目の場合は，マシンフレーム番号 0x1d01cf へのアクセスがあったことを表している．page_info はページの属性を表している．1 行目の場合は，PGT_l4 と表示されている．これはレベル 4 ページテーブルへのアクセスであるということを表している．同様に 2 行目の PGT_l3 はレベル 3 ページテーブル，3 行目の PGT_l2 はレベル 2 ページテーブル，4 行目の PGT_l1 はレベル 1 ページテーブルをそれぞれ表している．なお 5 行目のようにページの種類が不明であった場合は “Other” と表示される．

このログから 5 行で一組になっていることがわかる．ログの page_info の部分を見るとレベル 4〜1 までのページテーブルにアクセスした後に Other のページをマップしている．ページテーブルへのアクセスはアドレス変換を表し，管理 VM はアドレス変換を行うための API を実行したと考えられる．その後 Other のページへのアクセスは，アドレス変換によって求められたページへアクセスするためのメモリマップ用ハイパーコールによるものであると考えられる．実際にログに示されたマシンフレームの領域を確認したところ，ページテーブルや，プロセス構造体が含まれる領域であったことから，メモリアクセス履歴通知機構が動作していることが確認できた．

5.2 メモリアクセス制御機構の動作確認

メモリアクセス制御機構の動作確認を行うために，前節のメモリアクセス履歴通知機構の確認を行った直後に，ユーザ VM からメモリアクセス制御リストに禁止領域を指定し，ユーザ VM 内のプロセス一覧を取得するプログラムを管理 VM 上で実行した．なお，禁止領域として図 6 の 15 行目 (count:240) にあるマシンフレーム番号 0x1c5d60 をメモリアクセス制御リストに登録した．

管理 VM の出力結果を**図 7**，ユーザ VM への出力ログを**図 8** に示す．なお，図 7，図 8 ともに，図 5，図 6 と異なり，すべてのログを掲載している．

```
1 PID:[      1] COMM:init    PATH:/sbin/init
2 Failed to map guest memory
```

図 7 アクセス制御導入時の管理 VM の画面出力

```
1  [count:476 mfn:0x1d01cf page_info:PGT_l4]
2  [count:477 mfn:0x1fabb4 page_info:PGT_l3]
3  [count:478 mfn:0x1fabb0 page_info:PGT_l2]
4  [count:479 mfn:0x1f89be page_info:PGT_l1]
5  [count:480 mfn:0x1fabae page_info:Other ]
6  [count:481 mfn:0x1d01cf page_info:PGT_l4]
7  [count:482 mfn:0x1fabb5 page_info:PGT_l3]
8  [count:483 mfn:0x1fabb1 page_info:PGT_l2]
9  [count:484 mfn:0x1fa447 page_info:PGT_l1]
10 [count:485 mfn:0x1c5d61 page_info:Other ]
11 [count:486 mfn:0x1d01cf page_info:PGT_l4]
12 [count:487 mfn:0x1fabb5 page_info:PGT_l3]
13 [count:488 mfn:0x1fabb1 page_info:PGT_l2]
14 [count:489 mfn:0x1fa447 page_info:PGT_l1]
15 [count:490 mfn:0x1c5d60 page_info:Other ]
16 Warning! Admin peeks Keepout Area mfn:0x1c5d60
```

図 8 メモリアクセス制御導入時のログ

図 8 より，アクセス制御機構を導入することにより，プロセス一覧を管理 VM は取得できずに，メモリマップが失敗していることがわかる．また，図 8 の 16 行目から禁止領域へのアクセスがあったことがわかり，0x1c5d60 上にあるプロセス構造体へアクセスするためのメモリマップ用ハイパーコールが失敗した．これにより，以降のプロセス構造体へのアクセスが不可能になった結果，ログおよび，管理 VM の画面出力が途中で止まったと考えられる．なお，図 8 からは 2 つの task_struct 構造体が取得できていると考えられるが，図 7 では，1 プロセス分のプロセス情報しか表示されていない．これは管理 VM 上ではプロセス ID が 0 のプロセスの情報を表示していないためである．以上より，メモリアクセス制御機構が動作していることが確認できた．

5.3 サスペンド時間

ユーザ VM のサスペンドに要する時間を，提案システム未導入の無修正版 Xen の場合，提案システムを導入して，ユーザ VM メモリをすべてのページを暗号化した場合と，提案システムを導入して，アクセス制御によりプロセス構造体が含まれるページのみを暗号化した場合の 3 条件で測定した．なお，起動中のプロセス数は 55 プロセスである．

実行結果を**図 9** に示す．無修正版と比べて，すべて暗号化した場合は 170%の実行時間が増加し，プロセス構造体のみを暗号化した場合は 5%実行時間が増加した．提案システムのメモリアクセス制御により暗号化が必要なページが減ることでサスペンド時間が短くなることがわかった．

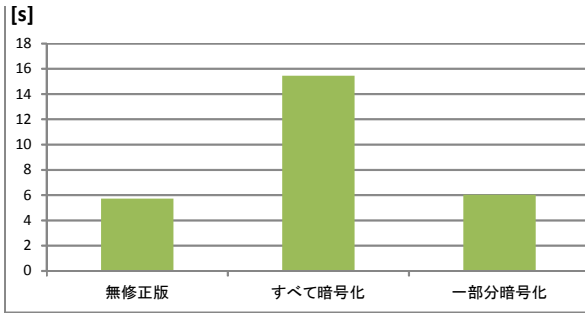

図 9 サスペンド時間

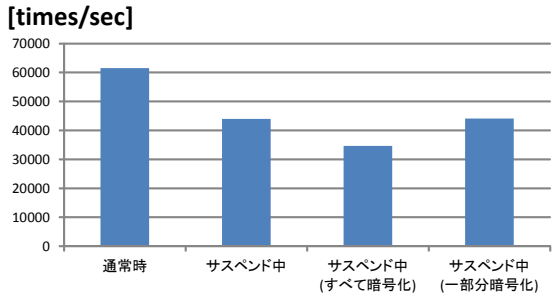

図 10 文字列出力プログラムの性能評価

5.4 VMM 負荷の影響

VMM がサスペンドや暗号化を行っている最中に，サスペンド対象ではない別の VM で文字列出力を繰り返すプログラムが 1 秒間に実行できる回数を測定した．実験環境は 2.2 節に示した通りである．さらに，実験条件として，提案システムを導入し，アクセス制御によりプロセス構造体が含まれるページのみを暗号化した場合を追加している．なおこの際動作しているプロセス数はサスペンド時間の実験と同様に 55 プロセスである．

実行結果を**図 10** に示す．暗号化無しのサスペンド中は通常時と比べて 29%，すべてのページを暗号化するサスペンド中は通常時と比べて 44%，プロセス構造体が含まれるページのみを暗号化するサスペンド中は通常時と比べて 28%の性能低下となった．暗号化無しとプロセス構造体のみを暗号化した場合の結果はほぼ同じとなり，暗号化を行うページがメモリ全体と比べて十分に少なければ，暗号化による性能低下はほとんど発生しないことがわかった．

6. まとめと今後の課題

本稿では IaaS 環境において管理 VM によるメモリ覗き見を監視するためのメモリアクセス監視機構を提案した．提案システムは，管理 VM がユーザ VM メモリへアクセスを VMM が検知し，VMM はメモリアクセス履歴をユーザ VM に通知する．さらに，ユーザ VM はメモリアクセス制御リストに管理 VM に覗かれたくない領域を指定可能であり，VMM はメモリアクセス制御リストに指定されている領域に対して管理 VM がアクセスを行った際に暗号化などを行うことでユーザ VM メモリを保護することができる．

今後の課題にユーザインタフェースの充実がある．現状では，ユーザ VM の仮想アドレスおよび，プロセス ID のみに対応しているが，より柔軟なセキュリティポリシを設定するにはこれには不十分であるため，より多くの形式で指定可能にする必要がある．また，通知ログの内容の充実も必要である．現状のログでは管理 VM がどのような領域にアクセスを試みたかを知ることは難しい．この問題を解決するために，ログを解析し，アクセスされた領域の詳細を取得する機能を開発する必要がある．

参考文献

[1] Nance, K., Bishop, M. and Hay, B.: Virtual Machine Introspection: Observation or Interference?, *IEEE Security and Privacy*, Vol. 6, pp. 32–37 (2008).

[2] Jiang, X., Wang, X. and Xu, D.: Stealthy malware detection through vmm-based ”out-of-the-box” semantic view reconstruction, *Proceedings of the 14th ACM conference on Computer and communications security*, CCS ’07, New York, NY, USA, ACM, pp. 128–138 (2007).

[3] Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *In Proc. Network and Distributed Systems Security Symposium*, pp. 191–206 (2003).

[4] Hidekazu, T., Kenichi, K. and Shigeru, C.: Preventing Information Leakage from Virtual Machines’ Memory in IaaS Clouds, 情報処理学会論文誌コンピューティングシステム（ACS）， Vol. 5, No. 4, pp. 101–111 (2012).

[5] Li, C., Raghunathan, A. and Jha, N. K.: A Trusted Virtual Machine in an Untrusted Management Environment, *IEEE Transactions on Services Computing*, Vol. 5, No. 4, pp. 472–483 (2012).

[6] G.Coker: Xen Security Modules (XSM). presented at the 2007 Xen Summit.

ⓒ 2013 Information Processing Society of Japan