Vol.2016-OS-137 No.12
2016/5/31

クラウドのリモート管理における VMリダイレクト攻撃の防止

猪口 恵介[1]　光来 健一[1]

概要：近年，利用が広まってきているIaaS型クラウドでは，ユーザは提供される仮想マシン（VM）を管理VMを経由して利用する．しかし，管理VMを管理するクラウド管理者は必ずしも信頼できるとは限らず，ユーザのVMに対して様々な攻撃を行う可能性がある．考えられる攻撃の一つとして，ユーザの接続するVMを変更する*VM*リダイレクト攻撃が考えられる．本稿では，VMのリモート管理においてVMリダイレクト攻撃を防止するシステム*UVBond*を提案する．UVBondは暗号化ディスクを介してハイパーバイザレベルでユーザとVMを強く結びつける．そして，ハイパーバイザが発行したセキュアなVM識別子を用いてユーザのVMへのアクセスを保証する．我々はUVBondをXenに実装し，ハイパーバイザで暗号化・復号化を行うことによって準仮想化ドライバを用いるVMを暗号化ディスクから起動できるようにした．その上で，発行されたVM識別子を用いた場合にのみ対応するVMが操作できるようにした．

1. はじめに

近年，急速にクラウドコンピューティングの普及が進んでいる．クラウドコンピューティングのサービス形態の一つであるIaaS型クラウドでは，ユーザに仮想マシン（VM）の提供を行う．ユーザがVMを管理する際には，まずネットワークを介して管理サーバにアクセスし，そこから管理VMと呼ばれるVMに接続した上でユーザVMへのアクセスを行う．例えば，ユーザは管理VMを通してユーザVMの起動や終了，マイグレーションなどの操作を行ったり，VNCやSSHなどのリモート管理システムを用いた帯域外リモート管理を行ったりする．

管理VMはクラウドのシステム管理者によって管理されているが，クラウド管理者は必ずしも信頼できるとは限らない．そのため，信頼できないクラウド管理者が管理VMの権限を悪用し，ユーザVMに対して様々な攻撃を行うことが考えられる．考えられる攻撃の一つとして，ユーザがアクセスするVMを意図的に変更し，管理VMによって用意された悪意あるVMに接続させる攻撃が考えられる．本稿ではこのような攻撃を*VM*リダイレクト攻撃と呼ぶ．VMリダイレクト攻撃によって悪意ある管理者が用意したVMにアクセスさせられた場合，VM内部にインストールされたマルウェアなどによってユーザ情報が盗まれる可能性がある．

本稿では，VMの暗号化ディスクを介してユーザとVMを強く結びつけることによりVMリダイレクト攻撃を防ぐ*UVBond*を提案する．UVBondはクラウド内のハイパーバイザだけを信頼し，ハイパーバイザレベルでのディスク暗号化を用いてユーザの指定したディスクイメージからVMを起動する．その際に自分のVMが起動されたことをユーザ自身が確認することを可能にする．VMが正しく起動されていればUVBondはユーザにセキュアなVM識別子を発行し，ユーザはVM識別子を用いてVMへのアクセスを行う．ハイパーバイザにおいてVM識別子に対応するVM以外へのアクセスを拒否することにより，管理VMにおいてアクセス先のVMが変更されることを防ぐ．

我々はXen 4.4.0 [1]にUVBondを実装した．UVBondではハイパーバイザにディスクの暗号鍵を登録し，準仮想化ドライバを用いるVMのディスクに対してハイパーバイザ内部で暗号化・復号化を実行する．さらに，VM識別子を要求するコマンドを作成することで，ハイパーバイザが管理VMによるVMリダイレクト攻撃を防止することを可能とする．UVBondにおいてVM識別子を用いた操作を行い，正しいVM識別子を指定した場合にのみVMが操作可能であることが確認できた．さらに，比較対象を含めた3種類のVMでディスクI/Oの性能を測定する実験を行った．実験の結果，UVBondでは暗号化を行わない通常のVMと比較して，読み込み性能は24%，書き込み性能は16%低下することが分かった．

以下，2章ではクラウドにおける管理VMの信頼性と

[1] 九州工業大学
Kyushu Institute of Technology

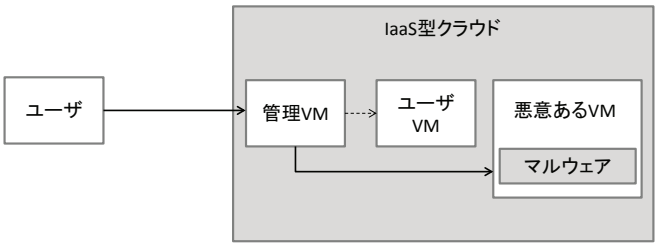

図 1　VM リダイレクト攻撃

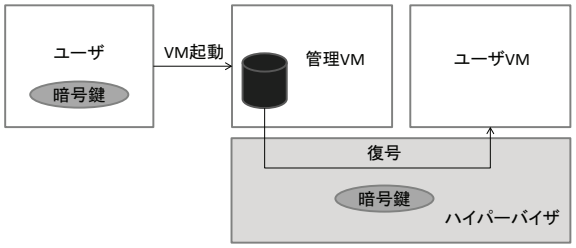

図 2　暗号化ディスクを用いた VM の起動

VM リダイレクト攻撃について述べ，3 章では提案するシステムである UVBond について述べる．4 章では実装の詳細について述べ，5 章では行った実験について述べる．6 章では関連研究に触れ，7 章ではまとめと今後の課題について述べる．

2. VM リダイレクト攻撃

クラウド内のユーザ VM は，各ホストで動作する管理 VM と呼ばれる特権を持った VM を経由して管理される．管理 VM はユーザ VM に対して起動や停止，バックアップの作成，他のホストへのマイグレーションなど様々な操作を行うことができる．また，管理 VM 内に作成されるユーザ VM の仮想デバイスに直接アクセスすることにより，ユーザ VM の帯域外リモート管理を行うこともできる．帯域外リモート管理はユーザ VM のネットワークに依存せずに SSH や VNC などを用いてユーザ VM 内のシステムにログインしてアクセスすることを可能にする．

管理 VM はクラウドのシステム管理者によって管理されているが，クラウドの管理者は必ずしも信頼できるとは限らない．例えば，Google の管理者がユーザのプライバシを侵害するという事件 [2] が発生している．また，サイバー犯罪の 28%は内部犯行 [3] であり，管理者の 35%は機密情報に無断でアクセスしたことがある [4] という報告もある．このように，クラウドサービスプロバイダ自体は信頼することができるとしても，クラウド内にその管理権限を悪用する管理者が存在しないことを保証するのは難しい．

信頼できないクラウド管理者は管理 VM の権限を悪用し，ユーザ VM に対して様々な攻撃を行う可能性がある．例えば，管理 VM からはユーザ VM のメモリや帯域外リモート管理の入出力の情報を盗み見ることができる．このような情報漏洩を防ぐために，ユーザ VM のメモリや入出力を暗号化する手法が提案されてきた [5], [6], [7], [8], [9]．これらの手法では，管理 VM やユーザ VM の下で動作するハイパーバイザを信頼することにより，管理 VM に対してユーザ VM の情報を隠蔽する．

本稿では新たな攻撃として VM リダイレクト攻撃を考える．VM リダイレクト攻撃は図 1 のように管理 VM においてユーザがアクセスする VM を変更する攻撃である．クラウドの管理者はマルウェアなどをインストールした悪意ある VM を作成しておき，ユーザをその VM にアクセスさせることによって様々な攻撃を行うことができる．例えば，システムのログインプログラムを改ざんしたり，キーロガーを仕掛けることによって，ユーザがシステムにログインする際にパスワードを盗むことができる．この攻撃はアクセス先の VM 内部で行われるため，ユーザ VM の帯域外リモート管理における入出力経路を暗号化したとしても防ぐことはできない．

3. UVBond

3.1 脅威モデル

本稿では，クラウド内に信頼できない管理者が存在する可能性があり，クラウド全体が信頼できるとは限らない状況を想定する．信頼できない管理者は管理 VM を用いてクラウド内のユーザ VM の管理を行うため，管理 VM は信頼することができないものとして考える．信頼できない管理 VM において行われる攻撃として，ユーザがアクセスする先の VM を変更される VM リダイレクト攻撃を想定する．

一方で，クラウドプロバイダは信頼できるものとし，仮想化システムの根幹となるハイパーバイザおよびその下のハードウェアは信頼できるものとする．このような仮定は様々な研究において行われている [7], [8], [9], [10], [11], [12]．信頼できるハイパーバイザは様々な手法によって実現可能である．例えば，TPM を用いたセキュアブートによってハイパーバイザが正常に起動したことを保証することができる．また，ハードウェアを用いた監視手法により，実行時の改ざんを検出することもできる [13], [14], [15], [16]．

3.2 UVBond

本稿では，VM の暗号化ディスクを介してユーザと VM を強く結びつけることによって VM リダイレクト攻撃を防ぐ UVBond を提案する．クラウドにおいて情報漏洩を防ぐには従来でもディスクを暗号化する必要があったため，UVBond においてディスク暗号化を用いてもオーバヘッドが大きく増大することはない．UVBond は従来の管理 VM

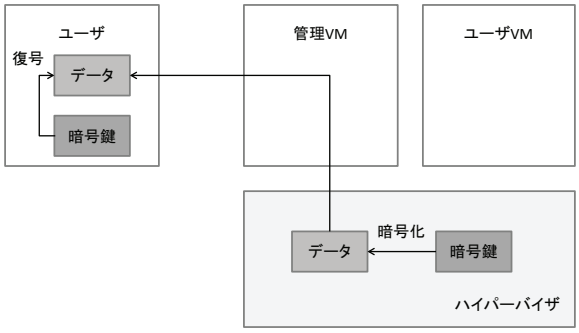

図 3　ディスクの正しさの確認

やユーザ VM 内でのディスク暗号化ではなく，ハイパーバイザレベルでのディスク暗号化を用いる．ユーザ VM の起動時に，ユーザとハイパーバイザはディスク暗号鍵を共有し，ハイパーバイザ内でユーザ VM とディスク暗号鍵を結びつける．そして，図 2 のようにディスクの読み書きの際にデータの暗号化・復号化を行う．

UVBond はユーザの所有する暗号化ディスクとディスク暗号鍵の組を用いて，正しくユーザ VM が起動されたかどうかの確認を行う．信頼できない管理者は不正な暗号化ディスクを作成し，そのディスク暗号鍵をハイパーバイザに登録してユーザ VM を起動する可能性があるためである．この確認を行うため，図 3 に示すようにハイパーバイザは登録されたディスク暗号鍵で暗号化したチェック用データをユーザに送信する．ユーザが受信したデータを正常に復号できれば，ハイパーバイザに登録されたディスク暗号鍵がユーザのものであることが確認できる．暗号化ディスクとディスク暗号化の組が対応していない場合にはユーザ VM が正常に起動しないため，ユーザの暗号化ディスクが用いられているかどうかは容易に判別できる．

ユーザ VM が起動された時に，ハイパーバイザはユーザ VM に結びつけられたセキュアな VM 識別子をユーザに対して発行する．VM 識別子は図 4 に示すように，ユーザが VM にアクセスする際に使用され，ユーザが指定した VM 以外にアクセスすることを防ぐ．そのため，信頼できない管理者が VM リダイレクト攻撃を行おうとしても，リダイレクト先の VM へのアクセスはハイパーバイザによって拒

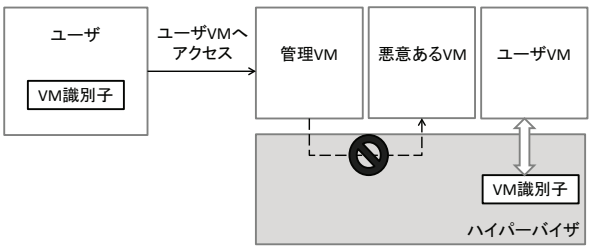

図 4　VM 識別子を用いた VM の操作

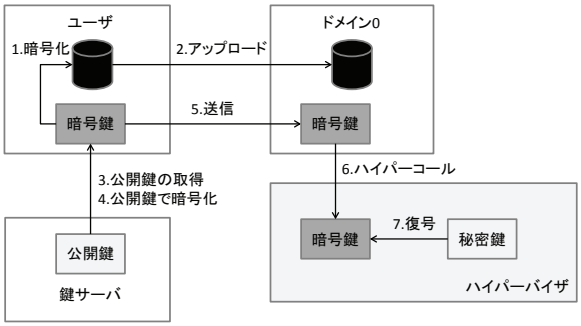

図 5　ディスクのアップロードと暗号鍵の登録

否される．

4. 実装

我々は，UVBond を Xen 4.4.0 [1] に実装した．ハイパーバイザ内での暗号化・復号化のために WolfSSL[17] の AES，RSA を移植し，ユーザ端末のクライアントでは OpenSSL を使用した．

4.1 VM のライフサイクル

UVBond を用いた場合の VM のライフサイクルは以下のようになる．ユーザは VM の作成時に AES 方式のディスク暗号鍵を生成し，この暗号鍵を用いて VM のディスクイメージを暗号化する．暗号化されたディスクイメージはクラウドにアップロードされ，従来通りドメイン 0 内に格納される．

VM を起動する際に，クライアントは VM の起動コマンドとともに，ハイパーバイザの RSA 公開鍵によって暗号化されたディスク暗号鍵をドメイン 0 に送信する．ハイパーバイザの公開鍵はクラウドプロバイダによって信頼できる鍵サーバに登録されていることを仮定する．クライアントは鍵サーバから公開鍵を取得すべきであるが，鍵サーバは現在未実装であるためあらかじめクライアントに登録している．暗号化されたディスク暗号鍵を受け取ったドメイン 0 はハイパーコールを発行して受信データをハイパーバイザに渡す．ハイパーバイザは自身の秘密鍵でこのデータを復号し，起動しようとしている VM にディスク暗号鍵を登録する．ディスクイメージの暗号化から暗号鍵の復号までのプロセスを図 5 に示す．

ドメイン 0 は起動コマンドに基づき，ユーザが指定した暗号化ディスクを用いて VM の起動を行う．VM がディスクにアクセスする際に，ハイパーバイザがディスクアクセスを横取りし，読み込もうとしているデータの復号，および書き込もうとしているデータの暗号化を行う．この処理の詳細については 4.2 節および 4.3 節で述べる．ハイパーバイザとクライアントは，VM の起動と並行して用いられている暗号化ディスクとディスク暗号鍵の正しさを確認する．この詳細については 4.4 節で述べる．

VM が起動すると，ドメイン 0 はハイパーコールを発行してハイパーバイザからセキュアな VM 識別子を取得する．この VM 識別子はハイパーバイザによって生成され，VM に登録されているディスク暗号鍵を用いて暗号化されている．ドメイン 0 が VM 識別子をクライアントに送信すると，クライアントはディスク暗号鍵を用いてそれを復号する．クライアントが VM に対して操作コマンドを実行する際にはこの VM 識別子を指定する．この詳細については 4.5 節で述べる．

4.2 準仮想化ディスク I/O の暗号化

VM のディスクアクセスには準仮想化ディスクドライバが用いられることが多い．完全仮想化ディスクドライバを用いる場合と比べて，ディスク入出力の性能を向上させることができるためである．Xen は準仮想化と完全仮想化の両方のゲスト OS をサポートしているが，完全仮想化の場合でも Linux では準仮想化ディスクドライバが用いられる．

4.2.1 従来のディスク入出力

Xen の準仮想化ディスクドライバは，ドメイン U で動作する blkfront ドライバとドメイン 0 で動作する blkback ドライバからなる．これらのドライバは I/O リングと呼ばれるデータ領域を共有し，イベントチャネルと呼ばれる機構を用いて通信を行う．VM がディスクの入出力を行う際には，blkfront ドライバが I/O リングに要求を書き込み，イベントを送信する．イベントを受け取った blkback ドライバはその要求を I/O リングから取り出し，VM のディスクイメージにアクセスすることで入出力を実現する．読み込みの場合には，要求で指定されたドメイン U のバッファ領域を共有し，ディスクイメージから読み込んだデータを書き込む．書き込みの場合は，指定されたバッファ領域のデータをディスクイメージに書き込む．入出力が完了すると，blkback ドライバは応答を I/O リングに書き込み，blkfront ドライバにイベントを送信する．

ドメイン 0 とドメイン U の間でのメモリ領域の共有には，グラントテーブルと呼ばれる機構が用いられる．ドメイン U はまず，グラントテーブルに共有したいメモリページを登録する．このページはグラントページと呼ばれる．グラントページにはグラント参照が割り当てられ，ドメイン 0 はグラント参照を指定して対応するページをマップして共有する．I/O リングが格納されているページのグラント参照は XenStore と呼ばれるドメイン 0 内のデータベース経由で blkfront ドライバから blkback ドライバに渡される．ディスクの読み書きに用いられるバッファ領域が格納されているページについては，I/O リングに書き込まれる要求を通してグラント参照が渡される．

4.2.2 共有データの二重化

ドメイン 0 に復号後のディスクデータを傍受されないようにするために，UVBond はバッファ領域を格納しているグラントページを二重化する．ドメイン U には従来通り，暗号化されていないページを提供するが，ドメイン 0 には暗号化されたページを提供する．ドメイン U に提供するページをゲストグラントページと呼び，ドメイン 0 に提供するページをシャドーグラントページと呼ぶ．一方，I/O リングを格納しているグラントページについても二重化は行うが，暗号化は行わない．この二重化は，ハイパーバイザが暗号処理を終えた要求だけを blkback ドライバに渡し，復号処理を終えた応答だけを blkfront ドライバに渡せるように同期をとるために行う．ドメイン U に提供する I/O リングをゲスト I/O リング，ドメイン 0 に提供する I/O リングをシャドー I/O リングと呼ぶ．

UVBond は従来との互換性を保つために，ドメイン U から渡されたグラント参照を用いて，ドメイン 0 が二重化されたグラントページにアクセスすることを可能にする．ドメイン U に対しては従来通り，グラント参照をゲストグラントページに対応させる．一方，ドメイン 0 に対しては，同じグラント参照をシャドーグラントページに対応させる．これにより，ドメイン 0 がグラント参照に対してマップ処理を行うと，ゲストグラントページの代わりにシャドーグラントページがマップされ，アンマップ処理を行うとシャドーグラントページがアンマップされる．ドメイン 0 からはゲストグラントページにアクセスできているように見えるが，実際にはシャドーグラントページにアクセスすることになる．

4.2.3 暗号化ディスクを用いた入出力

UVBond では，VM の起動時にハイパーバイザがドメイン U とドメイン 0 の間の通信を解析することによって I/O リングを特定し，シャドー I/O リングを作成する．blkfront ドライバは初期化時に，XenStore に I/O リングのグラント参照を登録する．この際に，XenStore リングに要求を書き込んでイベントを送るため，イベントを送信するハイパーコール内で XenStore リングから I/O リングのグラント参照を取得する．XenStore リングが格納されているページの情報は VM の起動時にハイパーバイザに通知される．そして，シャドー I/O リングを作成した後で，ゲスト I/O リングの中身をシャドー I/O リングにコピーする．blkfront ドライバが用いるイベントチャネルのポート番号も同様にして取得する．

blkfront ドライバから blkback ドライバに要求が送られた時，それがディスクへの書き込み要求であればハイパーバイザはデータの暗号化を行う．要求を送る際にはイベントを送信するためのハイパーコールが呼ばれるため，その中でゲスト I/O リングに書き込まれた要求を解析する．要求に含まれるグラント参照に対応するシャドーグラントページがまだなければ作成する．ディスクへの書き込み要求の場合には，図 6 のようにゲストグラントページ上の

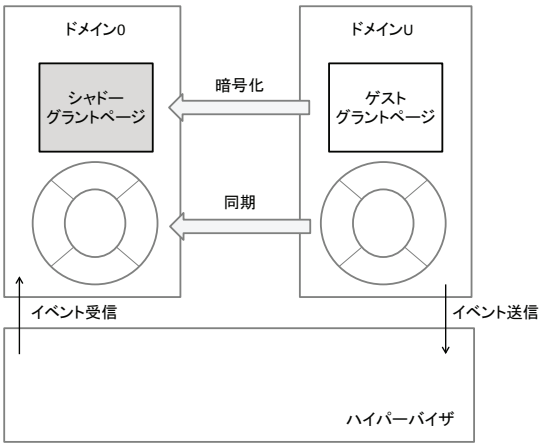

図 6　blkflont ドライバから blkback ドライバへの要求

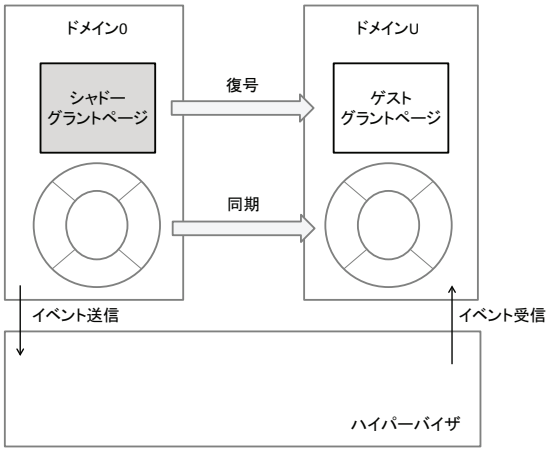

図 7　blkback ドライバから blkflont ドライバへの応答

バッファ領域に格納されたデータを暗号化しながらシャドーグラントページに書き込む．最後に，この要求をシャドー I/O リングにコピーする．

blkback ドライバから blkfront ドライバに応答が送られた時，それがディスクへの読み込み要求に対する応答であればハイパーバイザはデータの復号を行う．応答にはグラント参照が含まれないため，対応する要求を保存しておき，要求からグラント参照を取得する．そして，ディスクへの読み込み要求に対する応答の場合には，図 7 のようにグラント参照に対応するシャドーグラントページ上のバッファ領域に格納されたデータを復号しながらゲストグラントページに書き込む．最後に，この応答をゲスト I/O リングにコピーする．使用されなくなったゲストグラントページが解放された場合，同時に対応するシャドーグラントページも解放する．

4.3　完全仮想化ディスクの I/O の暗号化

UVBond は準仮想化ディスク I/O だけでなく，完全仮想化ディスク I/O にも対応している．これは，完全仮想化 OS を起動する場合，ゲスト OS が起動する前に実行される BIOS は完全仮想化ディスク I/O を用いるためである．BIOS からのディスク読み込みは，ハイパーバイザにおいて IN 命令をエミュレートすることで実現されており，512 バイト単位での読み込みが行われる．512 バイトの読み込みは 4 バイト分の IN 命令と 508 バイト分の IN 命令の繰り返しに分けてハイパーバイザにトラップされる．AES は 16 バイト単位でしか復号を行うことができないため，後者の 508 バイトの読み込み時に 512 バイト分をまとめてハイパーバイザ内で復号する．VM の起動時には BIOS のディスクへの書き込み機能は使われないため，書き込みの暗号化には未対応である．また，DMA 転送にも未対応である．

4.4　VM の正常起動の確認

VM の起動に用いられる暗号化ディスクとディスク暗号鍵が正しく対応していることを確認するために，UVBond はディスクのブートセクタのチェックを行う．ブートセクタは VM の起動時に必ず読み込まれ，固定のマジックナンバが格納されている．そのため，ハイパーバイザが復号後にマジックナンバを確認できれば，暗号化ディスクの単純な置き換えを自動的に検出することができる．VM が正しく起動することを完全に保証するには，マークルハッシュ木などを用いてディスク全体の整合性をチェックする必要がある．しかし，ハッシュデータがディスクイメージとは別に必要となり，ハッシュのデータ量が大きくなるためハイパーバイザ内に登録して用いるのは現実的ではない．一般に，このような整合性チェックを行わなくても，VM が正しく起動したかどうかはユーザが容易に判断することができる．

次に，UVBond はハイパーバイザに登録されたディスク暗号鍵がユーザのものであるかどうかの確認を行う．VM が正しく起動していたとしても，不正な暗号化ディスクとディスク暗号鍵の組を用いて起動されている可能性があるためである．起動した VM の VM 識別子を取得するためにドメイン 0 がハイパーコールを発行した際に，ハイパーバイザは VM 識別子にマジックナンバを付加してディスク暗号鍵で暗号化する．これを受け取ったクライアントは自身の持つディスク暗号鍵でこれを復号し，マジックナンバが正しく復号できればユーザの持つ鍵とハイパーバイザに登録されている鍵が同じものであることが確認できる．この確認に成功すれば以降の VM の操作には送られてきた VM 識別子を用いる．

4.5　VM 識別子を用いた VM 管理

クライアントは VM の管理を行う際に，VM 識別子とコマンドをディスク暗号鍵を用いて暗号化してドメイン 0 に送信する．ドメイン 0 がハイパーコールを発行してこれをハイパーバイザに渡すと，ハイパーバイザはディスク暗号鍵を用いて復号を行う．VM 識別子が操作を行おうとして

いる VM に対応するものであれば，指定されたコマンドを実行する．そうでなければ，VM リダイレクト攻撃とみなしてエラーを返す．現在のところ，VM 識別子を用いて VM の一時停止および再開を行うハイパーコールをサポートしている．

ドメイン 0 が暗号化されたコマンドを再利用し，ユーザが意図しないタイミングで VM への操作を行うことを防ぐために，UVBond はコマンドにカウンタ値も付加する．カウンタ値はクライアントがコマンドを送信する際に VM 識別子とともに暗号化され，ハイパーバイザに送信される．カウンタ値は操作が行われるたびにインクリメントするため，ドメイン 0 がコマンドを再利用しようとしても，カウンタ値を比較することで検出することができる．

5. 実験

UVBond の有効性を確かめるための実験を行った．実験には，Intel Xeon E3-1290 の CPU，8GB のメモリ，1TB の HDD を搭載したマシンを使用し，ハイパーバイザとして Xen 4.4.0 を用いた．管理 VM では Linux 3.16，ユーザ VM では Linux 3.13 を動作させた．ユーザ VM には 2 個の CPU と 2GB のメモリを割り当てた．

5.1 VM 識別子を用いた VM 管理

管理 VM による VM リダイレクト攻撃を防げることを確認するために，ユーザ VM に対して正しい VM 識別子を指定した場合にのみ VM が操作できることを確かめる実験を行った．そのために，管理対象のユーザ VM に対応する VM 識別子を用いた場合と不正な VM 識別子を用いた場合について，VM の一時停止と再開を行うコマンドを実行した．また，正しい VM 識別子を用いた場合には，正常なカウンタ値と不正なカウンタ値のそれぞれについて実験を行った．実験の結果，正しい VM 識別子および正常なカウンタ値を指定した場合には VM が正常に操作できることを確認した．一方，不正な VM 識別子または不正なカウンタ値を指定した場合には，VM への操作ができないことも確認した．

5.2 ディスク I/O 性能

UVBond が提供するディスク暗号化を用いるユーザ VM に対して，ディスク I/O ベンチマークツールである bonnie++を用いて性能測定を行った．ディスク I/O の性能測定には，比較対象も含め 3 種類の VM を用いて行った．用意した VM はそれぞれ，UVBond を用いてハイパーバイザレベルで暗号化を行う VM，暗号化を行わない通常 VM，dm-crypt を用いてゲスト OS レベルで暗号化を行う VM である．UVBond では CPU による暗号化サポートである AES-NI に対応できていないため，dm-crypt でも AES-NI を使わないようにして比較を行った．また，ディスクアクセスの際にドメイン 0 のページキャッシュ上のデータが使われると正確なディスク I/O 性能が得られないため，ドメイン 0 のメモリサイズを小さくして実験を行った．

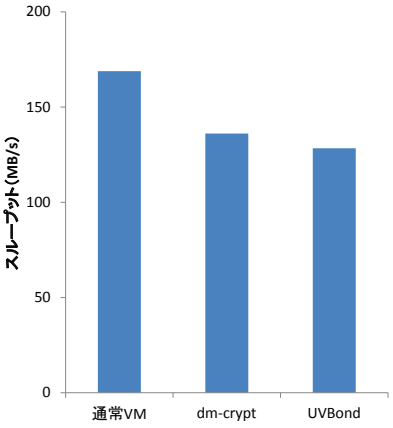

図 8 ディスクの読み込み性能

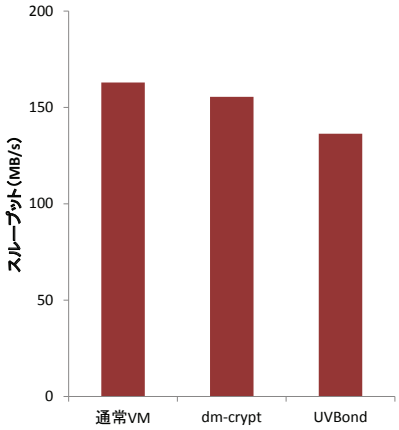

図 9 ディスクの書き込み性能

それぞれの VM において 10 回ずつ計測した読み込み性能と書き込み性能の平均値をそれぞれ図 8，図 9 に示す．読み込み性能に関しては，通常 VM と比較して dm-crypt は 19%の性能低下，UVBond は 24%の性能低下となった．dm-crypt は暗号化のオーバヘッドが影響していると考えられ，UVBond では暗号化のオーバヘッドに加えて管理 VM とユーザ VM 間の通信解析のオーバヘッドが影響していると考えられる．一方，書き込み性能は通常 VM と比較して dm-crypt は 5%の性能低下，UVBond は 16%の性能低下となっている．書き込み性能に関しても，読み込みの場合と同じ原因が考えられる．

6. 関連研究

Self-Service Cloud（SSC）[10] は，クラウドの管理者が干渉できないユーザ専用の管理 VM（Udom0）を提供するため，クラウド管理者は VM リダイレクト攻撃を行うことができない．Udom0 のディスクの整合性は vTPM を用

いて検査され，クライアントとの間は SSL で安全に接続される．ユーザは Udom0 を経由して自身の VM を安全に管理することができる．サービスドメインと呼ばれる VM を用いて，安全にディスクの暗号化を行うことも可能である．しかし，vTPM は DomB と呼ばれる VM で動作するため，ハイパーバイザに加えて DomB も信頼する必要がある．また，SSC ではクラウド管理者がユーザ VM をまったく管理できなくなる．UVBond ではクラウド管理者がユーザ VM を管理することも可能である．

BitVisor[18] はハイパーバイザ内の準パススルードライバを用いてディスクの暗号化を行うことができる．BitVisor はディスク暗号化に必要な最小限のハードウェアアクセスのみを横取りし，他のアクセスはパススルーすることができるため軽量な VM を実現できる．ATA ホストコントローラ用ドライバでは PIO 転送と DMA 転送に対応しており，シャドー DMA 記述子やシャドーバッファを用いて DMA 転送時の暗号化を実現している．UVBond と異なり，完全仮想化 OS にのみ対応しており，準仮想化ディスクドライバを用いることはできない．

CloudVisor[11] はネストした仮想化を用いてハイパーバイザの下で動作するセキュリティモニタにおいて，ディスクの暗号化および整合性チェックを行う．整合性チェックに必要なハッシュデータは管理 VM 経由で提供される．これにより，ユーザが指定したディスクイメージを用いて VM が正常に起動されることが保証される．しかし，ディスク暗号化は外部からの VM のリモート管理とは結びつけられていないため，管理 VM において VM リダイレクト攻撃を行うことが可能である．

FBCrypt[5] は帯域外リモート管理における入出力をハイパーバイザレベルで暗号化することで管理 VM への情報漏洩を防ぐ．UVBond と同様に，FBCrypt はフレームバッファを二重化し，暗号化されたフレームバッファをドメイン 0 に提供する．ただし，FBCrypt はページフレーム番号を用いたメモリ共有を仮定している．また，FBCrypt では I/O リングの二重化を行っていない．キーボード入力については，ハイパーコールを用いて I/O リングへの書き込みを行うようにすることで，入力が復号される前にゲスト OS に渡されるのを防いでいる．そのため，バックエンドドライバへの修正が必要である．一方，画面出力についてはユーザに少し古い画面を見せる可能性がある．UVBond では I/O リングを二重化することで，ディスクドライバを修正することなく同期の問題を解決している．

7. まとめ

本研究では，VM の暗号化ディスクを介してユーザと VM を強く結びつけることによって VM リダイレクト攻撃を防ぐ UVBond を提案した．UVBond ではハイパーバイザレベルでディスクの暗号化・復号化を行うことでユーザの VM が正しく起動されたことをユーザ自身が確認することができる．ユーザはハイパーバイザが発行したセキュアな VM 識別子を用いることでアクセス先の VM が変更された場合にはそのことを検出することができる．

今後の課題として，現在の実装では単一 VM にしか対応していないため，複数 VM に対応することが挙げられる．また，UVBond を用いることによる性能低下が大きいため，AES-NI に対応することなどによる性能の向上も今後の課題である．AES-XTS のような実用的な暗号アルゴリズムにも対応する必要がある．さらに，VM をマイグレーションした後でも同じ VM 識別子を用いることができるようにすることも計画している．

参考文献

[1] 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).
[2] TechSpot News: Google Fired Employees for Breaching User Privacy, http://www.techspot.com/news/40280-google-fired-employees-for-breaching-user-privacy.html (2010).
[3] PwC: US Cybercrime: Rising Risks, Reduced Readiness (2014).
[4] CyberArk Software: Global IT Security Service (2009).
[5] Egawa, T., Nishimura, N. and Kourai, K.: Dependable and Secure Remote Management in IaaS Clouds, *Proc. Int. Conf. Cloud Computing Technology and Science* (2012).
[6] Kourai, K. and Kajiwara, T.: Secure Out-of-band Remote Management Using Encrypted Virtual Serial Consoles in IaaS Clouds, *Proc. Int. Conf. Trust, Security and Privacy in Computing and Communications*, pp. 443–450 (2015).
[7] Li, C., Raghunathan, A. and Jha, N. K.: Secure Virtual Machine Execution under an Untrusted Management OS, *Proc. Int. Conf. Cloud Computing*, pp. 172–179 (2010).
[8] Li, C., Raghunathan, A. and Jha, N. K.: A Trusted Virtual Machine in an Untrusted Management Environment, *IEEE Trans. Services Computing*, Vol. 5, No. 4, pp. 472–483 (2012).
[9] Tadokoro, H., Kourai, K. and Chiba, S.: Preventing Information Leakage from Virtual Machines' Memory in IaaS Clouds, *IPSJ Online Trans.*, Vol. 5, pp. 156–166 (2012).
[10] Butt, S., Lagar-Cavilla, H. A., Srivastava, A. and Ganapathy, V.: Self-service Cloud Computing, *Proc. Conf. Computer and Communications Security*, pp. 253–264 (2012).
[11] Zhang, F., Chen, J., Chen, H. and Zang, B.: CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization, *Proc. Symp. Operating Systems Principles*, pp. 203–216 (2011).
[12] Santos, N., Gummadi, K. P. and Rodrigues, R.: Towards Trusted Cloud Computing, *Proc. Workshop Hot Topics in Cloud Computing* (2009).
[13] Rutkowska, J. and Wojtczuk, R.: Preventing and Detecting Xen Hyper-visor Subversions, *Black Hat USA* (2008).

[14] McCune, J., Parno, B., Perrig, A., Reiter, M. and Isozaki, H.: Flicker: An Execution Infrastructure for TCB Minimization, *Proc. European Conf. Computer Systems*, pp. 315–328 (2008).
[15] Wang, J., Stavrou, A. and Ghosh, A.: HyperCheck: A Hardware-Assisted Integrity Monitor, *Proc. Int. Symp. Recent Advances in Intrusion Detection*, pp. 158–177 (2010).
[16] Azab, A., Ning, P., Wang, Z., Jiang, X., Zhang, X. and Skalsky, N.: HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity, *Proc. Conf. Computer and Communications Security*, pp. 38–49 (2010).
[17] wolfSSL Inc.: wolfSSL — Embedded SSL Library for Applications, Devices, IoT, and the Cloud, http://www.wolfssl.com/.
[18] T. Shinagawa and H. Eiraku and K. Tanimoto and K. Omote and S. Hasegawa and T. Horie and M. Hirano and K. Kourai and Y. Oyama and E. Kawai and K. Kono and S. Chiba and Y. Shinjo and K. Kato: BitVisor: A Thin Hypervisor for Enforcing I/O Device Security, *Proc. Int. Conf. Virtual Execution Environments*, pp. 121–130 (2009).