情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2009-ARC-183 No.10
Vol.2009-OS-111 No.10
2009/4/22

仮想マシン間における covert timing channel の評価

岡 村 圭 祐[†1] 大 山 恵 弘[†1]

仮想マシンモニタ上で実行されるそれぞれの仮想マシンは，同一仮想マシンモニタ上の他の仮想マシンから干渉を受けないように隔離されることで，セキュリティが高められている．ところが，実 CPU を covert channel として用いることによる通信の可能性があり，この通信によってセキュリティが脅かされる危険性が存在する．しかし，その危険性の大きさについては明らかではない．そこで本研究では，実 CPU を covert channel として用い，単位時間当たりの計算量を元に仮想マシン間で通信を行うシステムを構築した．このシステムは，Xen ハイパーバイザ上の 2 つのドメインで実行され，一方が送信側，他方が受信側として動作する．また，それぞれのドメイン内では，ユーザプロセスとして実行される．このシステムを用いて，実 CPU による covert channel の通信路としての性能を評価し，危険性の大きさを明らかにした．評価の結果，仮想マシンモニタを用いたコンピュータシステムのセキュリティを考える上で，無視することのできない脅威があることが確認できた．

Evaluation of Covert Timing Channels between Virtual Machines

Keisuke Okamura[†1] and Yoshihiro Oyama[†1]

Multiple virtual machines on a single virtual machine monitor are isolated from each other. A malicious user in one virtual machine usually cannot leak out secret data to other virtual machines without using explicit communication media such as shared files or a network. However, the isolation is threatened by communication in which a CPU load is used as a covert channel. Unfortunately, the threat of the covert channel has not been fully understood or evaluated. In this work, we quantitatively evaluate the threat of CPU-based covert channels between virtual machines on the Xen hypervisor. We have developed a system that creates a covert channel and communicates data secretly using CPU loads. The system consists of two user processes: a sender and a receiver. The sender runs in one virtual machine and the receiver runs in another virtual machine on the same hypervisor. We measured the bandwidth and precision of the covert channel and clarified the amount of security risk. We found out that there exists a threat to the isolation provided by the Xen hypervisor.

1. はじめに

近年，仮想専用サーバ (VPS) 方式でホスティングサービスを行っている企業が登場している．このサービスは，ある物理的なマシン上に存在する資源を仮想マシン単位で領域分割し，その仮想マシンを各ユーザにレンタルするものである．このとき，各仮想マシンのユーザが異なるという状態にあるので，別の仮想マシンに情報が漏出しないように注意を払う必要がある．一般的に，各仮想マシン同士は論理的に互いに独立しており，影響を及ぼしあうことはない．

通常，仮想マシン間で通信を行うには，通信を行うために意図された，ネットワークなどの正規の媒体 “overt channel” を使う必要がある．そのため，ある仮想マシンの信頼できないプログラムが他の仮想マシンに機密データを送信しようとしている場合，セキュリティソフトウェアなどを用いて overt channel を監視することで，その送信を阻止することができる．しかし従来から，通信路として意図されていない媒体，すなわち covert channel[1)2)] を用いた通信の可能性が指摘されている．この媒体により通信を行われると，ネットワークのような overt channel だけ監視を行っていても，情報の漏出を完全に防ぐことができなくなる問題が存在している．

計算機内のすべての共有資源は，covert channel として悪用される危険性がある．その代表的な共有資源として，実 CPU が挙げられる．特にシングルプロセッサの場合，計算機内のすべての仮想マシン，プロセスが共有することになる．つまり，同一の物理マシン上にある他の仮想マシンに，機密情報を収集して送信するスパイウェアを潜り込ませることができれば，データの通信路として実 CPU を用いて秘密裏に通信を行うことで，監視をすり抜けて情報を入手できる．この実 CPU を covert channel として用いた通信が，より大きなバンド幅で，よりよい精度であるほど，その脅威は大きくなる．しかしながら，この脅威の大きさは明らかではない．

そこで我々は，仮想マシン間で covert channel 通信を行うシステム CCCV (Covert Channels using CPU load between Virtual machines) を構築した．通信の媒体には，実 CPU の処理時間の差を用いた．これは，とりわけ covert timing channel と呼ばれる隠し媒体で

†1 電気通信大学
The University of Electro-Communications

© 2009 Information Processing Society of Japan

Vol.2009-ARC-183 No.10
Vol.2009-OS-111 No.10
2009/4/22

ある．この通信の評価を行うことで，仮想マシン間にまたがる covert channel がどの程度深刻な脅威となりうるのか指摘することを本研究の目的とする．このため，通信の精度やバンド幅について詳しく調査を行う．仮想マシンを提供するハイパーバイザとしては Xen[3] を用い，ハイパーバイザやゲスト OS のカーネルに手を加えずに通信を行うことを目標とする．

2. Xen

Xen では，仮想マシンをドメインと呼ばれる単位で管理している．ドメインは複数存在でき，それぞれのドメイン内に 1 つのゲスト OS が動作する．また，ドメイン 0 と呼ばれる特権ドメインが存在し，これが他のドメイン U と呼ばれる非特権ドメインを管理する権限をもつ．

Xen ハイパーバイザには，実行するゲスト OS すべてが所定の CPU 時間を受け取れるように保証する責任がある．そこでハイパーバイザ内には，各ドメインが持つ仮想 CPU (VCPU) と，物理 CPU の割り当てのスケジューリングを行うドメインスケジューラが存在する．このドメインスケジューラが各ドメインに CPU 時間を配分し，さらに各ドメイン内のゲスト OS のプロセススケジューラが，それぞれのアプリケーション (プロセス) へと CPU 時間を配分する．

Xen の現行スケジューラは，SEDF とクレジットスケジューラの 2 種類存在するが，最近のバージョンの Xen では，クレジットスケジューラが標準となっており，本研究でもこちらのスケジューラを対象としている．

クレジットスケジューラは，各ドメインにそれぞれ優先度と上限を設定する．ドメインが受け取る物理 CPU 時間は，優先度によって決定される．また，上限によって受け取ることのできる CPU 時間の最大値が設定される．このスケジューラは，それぞれのドメインの優先度から算出される “クレジット” (使用権が与えられる割り当て量) を定期的に配分し，CPU を利用した分だけそのドメインのクレジットを差し引いていく．このクレジットを使い切ってない状態をアンダースケジュール，使い切ってしまった状態をオーバースケジュールであるとして扱う．またこのスケジューラは，アンダースケジュール状態のドメインを優先し，これらのドメインをラウンドロビン方式でスケジューリングする．オーバースケジュール状態のドメインは，他のアンダースケジュール状態のドメインがアイドル状態の場合のみスケジューリングされる．このとき，足りなくなったクレジットは次に配分されるクレジットから差し引かれる．

3. 提案する通信システム CCCV

3.1 通信の概要

実 CPU を covert channel として用い，Xen 上のドメイン U (domU) 間で通信を行うシステム CCCV を構築した．CCCV は同じ実 CPU 上で動作する仮想マシン間で 10 進数 16 桁の数字列を通信することができる．この数字列の長さは，クレジットカード番号を漏えいさせる攻撃を想定して設定されている．

3.1.1 Covert channel としての実 CPU

実 CPU は本来，ハイパーバイザによって各ドメインから隠蔽されている．各ドメインにはそれぞれ VCPU が割り当てられていて，あたかも自ドメインが占有しているかのようにみえる．しかし実際には，あるドメインが実 CPU にスケジューリングされている間，他のドメインは待機することになる．ここで，待機中のドメインがあるタスクを実行中であったとすると，そのタスクの処理開始から終了までの経過時間には，ドメインが待機していた時間も含まれることになる．

これを踏まえて，実 CPU を covert channel として用いる方法を例を挙げて示す．今，ハイパーバイザ上にある domU はドメイン α とドメイン β の 2 つであり，これらのドメインは，処理開始から終了までに 4 単位時間分の CPU 時間が必要なタスク t を実行することができると仮定する．このときドメインスケジューラは，実行可能なタスクを持つドメインをラウンドロビン方式でスケジューリングするものとする．また，説明の簡略化のために，タスク t 以外の CPU 使用時間は 0 とし，ドメインスケジューラは 1 単位時間ごとにスケジューリングを行うものとする．

図 1 は，ドメイン α がタスク t を実行する際に，ドメイン β から受ける時間的影響を表したものである．図中の上段は，ドメイン β がタスク t を実行せず，アイドル状態である場合を示している．このとき，ドメイン α は実 CPU を独占的に使用することができ，タスク t の開始から終了までの経過時間は 4 単位時間となる．対して，図中の下段は，ドメイン α とドメイン β がほぼ同時にそれぞれタスク t を実行する場合を示したものである．この図では，ドメイン α が先にスケジューリングされた例を示している．このときドメイン α は，ドメイン β と 1 単位時間ごとに交互にスケジューリングを受けることとなり，一方がスケジューリングされている間は，他方が待機することになる．そのため，ドメイン α で実行しているタスク t の開始から終了までの経過時間は，7 単位時間に延びることになる．

この結果はつまり，ドメイン α はタスク t の開始から終了までの経過時間を測定するこ

© 2009 Information Processing Society of Japan

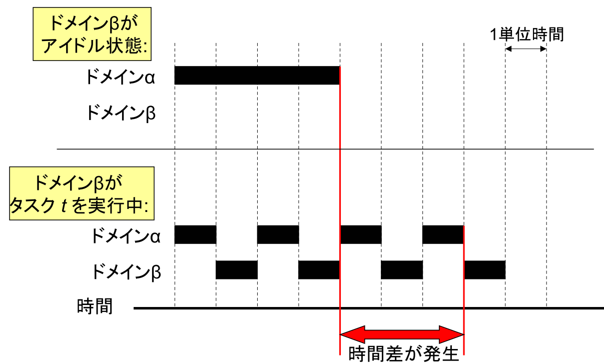

図 1　実 CPU の共有による計算時間の変化

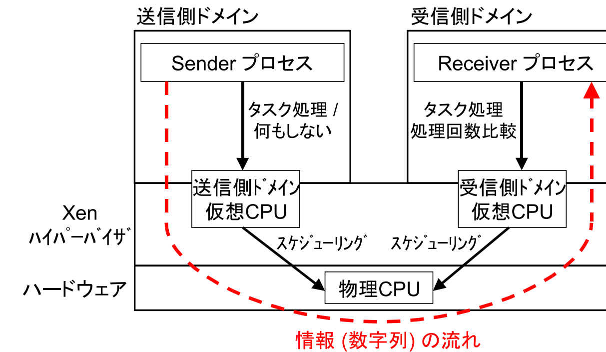

図 2　通信システムの概要図

とによって，その経過時間の長短から，ドメイン β がタスク t を実行していたか否かを推定できることを示している．そこで，ドメイン β を送信側，ドメイン α を受信側とし，あらかじめこの 2 つのドメイン間で「ドメイン β がタスク t を実行していたらビット 1，そうでなければビット 0」などの合意を定めておけば，ドメイン β からドメイン α に情報を送信することができる．

3.1.2　CCCV の概要

CCCV は，CPU 負荷を調節し情報を送信する sender プロセスと，その CPU 負荷を検出し情報を受信する receiver プロセスの 2 つで構成され，sender プロセスは送信側となるドメイン内で，receiver プロセスは受信側となるドメイン内で動作する (**図 2**)．これら 2 つのプロセスは，それぞれのゲスト OS のユーザプロセスとして動作する．また本システムは，10 進数 16 桁の数字列を通信する際，それぞれの桁を二進化十進表現 (BCD) を用いて表現している．よって，64 ビットの情報を送る必要がある．この情報は，1 ビットの通信方法を規定し，1 ビットの通信を 64 回繰り返すことで送信できる．そこで，1 ビットの通信方法について以下に述べる．

まず receiver プロセスは，sender プロセスが CPU に負荷をかけるような処理を何も実行していないときに，CPU 負荷がかかるようなあるタスクを一定時間で何回処理できるか測定しておき，これを基準処理回数として記憶しておく．続いて，sender プロセスと receiver プロセス間において，通信開始の同期をとる．同期の取り方については，3.2 節にて述べる．そののち，receiver プロセスは基準処理回数を測定したときと同じ時間中，同じタスクの処理を繰り返し行い，それが何回処理できるか測定する．同時に，sender プロセスはビット 1 を送信したい場合，同じタスク処理を一定時間繰り返し行い，CPU に負荷をかける．ビット 0 を送信したい場合には，引き続き処理を行わず CPU に負荷をかけない．最後に receiver プロセスは，通信時に処理できたタスクの回数と，基準処理回数の比較をする (**図 3**)．他のドメインによる CPU 負荷によって自ドメインのタスク処理の時間が延びることは 3.1.1 節にて述べたが，これは，一定時間中に処理できるタスクの回数が減少する，と言い換えることができる．よって，通信中に処理したタスクの回数と基準処理回数の差が大きいとき，ビット 1 を受信したとみなす．反対に，処理回数の差が小さかったとき，ビット 0 を受信したとみなす．

この 1 ビットの通信方法を踏まえて本システムを俯瞰すると，以下のようにまとめられる．

- sender プロセスは，10 進数 16 桁の数字列を表現する 64 ビットのビット列に基づいて CPU 負荷のかけ方を変化させる．
- receiver プロセスは通信中，64 回のタスク処理回数比較を通して 64 ビットのビット列を推定し，それを 10 進数 16 桁の数字列に解釈する．

また本システムは，送信側と受信側以外のドメインの処理が少ない (他のドメインによる CPU 使用率が 10%以下くらいが目安) と考えられる早朝などの時間帯に用いられることを想定し，通信開始から通信終了までの間に，他のドメインの CPU 使用状況が大きく変動しないことを前提とする．これは，タスク処理の遅延がどのドメインによってもたらされたものであるかを判別することが，本研究の covert channel の特性上，原理的に不可能である

© 2009 Information Processing Society of Japan

Vol.2009-ARC-183 No.10
Vol.2009-OS-111 No.10
2009/4/22

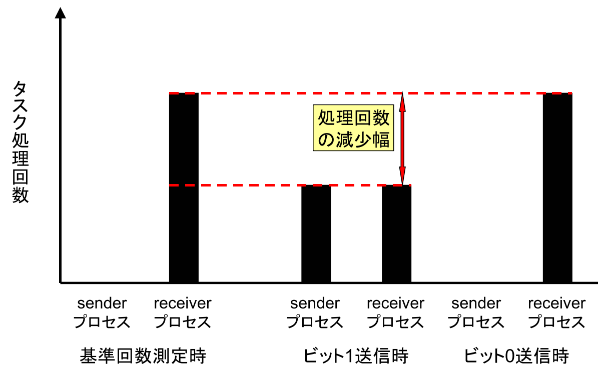

図 3 タスク処理回数の比較

からである．この前提が崩れ，2 つのドメイン間で通信中に他のドメインが大きな処理を開始した場合，受信側ドメインは発生した遅延が，送信側ドメインによって引き起こされたものか，処理を開始した第三者のドメインによって引き起こされたものか，判別ができなくなり，通信に混乱をきたす．また，通信開始前においては，通信開始の合図となる CPU 負荷のパターンを工夫し，精度の向上を試みる．詳しくは 3.2 節にて述べる．

通信を行うにあたって，送信側ドメインと受信側ドメインが実 CPU を共有している必要があるが，現状のシステムでは，特にシングルコア環境での通信を想定しており，マルチコア環境は対象外となる．

3.2 通信プロトコル

通信を精度よく効率的に行うために，独自のプロトコルを作成した．本システムはこのプロトコルに基づいて動作するものとする．

3.2.1 通信開始処理

本システムでは，sender，receiver の各プロセスがタイミングを合わせてタスク処理を実行することが非常に重要である．そこで両プロセスは，ビット列の通信を始める前に，通信に用いる CPU 負荷のタイミングの同期を取る必要がある．これを怠ると，両プロセスのビット通信過程にズレが発生する．このズレとは例えば，sender プロセスは既に 2 ビット目の送信処理を開始しているのだが，receiver プロセスは依然として 1 ビット目の受信処理を実行中である場合などが挙げられる．この場合，receiver プロセスに 1 ビット目と解釈される部分の CPU 負荷状況に，sender の意図とは異なる負荷状況が混じる可能性があり，それによって通信が乱されることになる．

この同期は，sender プロセスが特定の負荷パターンをかけ，receiver プロセスがその負荷パターンを検出することで実現できる．sender プロセスは通信を開始したいとき，1.5 秒の間タスク処理を繰り返し，CPU に負荷をかけたのち，次の処理へと移行するようにする．対して receiver プロセスは，約 0.1 秒ほど CPU 時間を必要とするタスク処理を繰り返し実行しながら待機している．CPU への負荷が検出されてから負荷が解消されるまでの時間が 1.5 秒付近であった場合，sender プロセスから受信要求が出されているとみなし，次の処理へと移行する．これにより，receiver プロセスは sender プロセスの送信手順に同期したタイミングで受信手順を実行できるようになる．receiver プロセスの処理は複雑であるので，詳しいアルゴリズムをアルゴリズム 1 に示す.「タスク処理時間計測」とは，1 つのタスクを実行し，そのタスク処理にかかった経過時間を測定することを表す．

アルゴリズム 1 について順を追って説明する．2–6 行目については，通信が開始される前に行う初期化処理を示している．この初期化処理として，receiver プロセスは約 0.1 秒ほど CPU 時間を必要とするタスクを，逐次的かつ連続的に 100 回実行する．このとき，100 回分のそれぞれのタスク処理にかかった経過時間を保持しておき，この平均値を求めておく．これは，通信に関係のない他のドメインによる CPU 使用状況を推定し，言わば他のドメインの CPU 使用量を差し引くために必要になる．初期化ののち，receiver プロセスは sender プロセスの受信要求を検出するため，引き続き同じタスクを逐次的かつ連続的に実行し続ける．8 行目以降は，この受信要求検出について示している．ここでは，先ほどの初期化処理にて得られた平均処理時間より 0.03 秒以上長い処理時間を必要としたタスクが，約 1.5 秒間だけ連続して発生した場合，sender プロセスの受信要求であると判断し，次の処理へと移る．この判断は，sender プロセスが発生させる CPU 負荷が 1.5 秒であるので，1.5 秒だけの間処理が遅延したタスクが発生し続け，その後すぐに平均処理時間付近で処理を行えるよう性能が回復したのは，sender プロセスによって引き起こされたものであると十分期待できることに基づいている．

また，この受信要求を行うことによって，sender プロセスと receiver プロセスのそれぞれの次の処理 (ビット列送受信) の同期をとることができる．これは，sender プロセスは受信要求を送信した後，調整のために 1 秒間のスリープを行い次の処理に移ることで，receiver プロセスは 1.5 秒間の遅延が終了した後，調整のために sender プロセスより少し短いスリープを行い次の処理に移行することで，自然と実現される．**図 4** は，これを模式的に表したものであり，矢印の根元がタスク処理開始時刻を，矢印の先がタスク処理終了時刻をそれぞ

© 2009 Information Processing Society of Japan

Vol.2009-ARC-183 No.10
Vol.2009-OS-111 No.10
2009/4/22

アルゴリズム 1 receiver プロセスの通信開始処理

```
1: { 初期化処理開始 }
2: while 反復回数 < 100 do
3:   タスク処理時間計測
4:   処理時間を記録
5: end while
6: ave = タスク処理時間の平均値
7: { 初期化処理終了 受信要求検出開始 }
8: repeat
9:   start_time ← タスク開始直前の時刻
10:  タスク処理時間計測
11:  if タスク処理時間 >= ave +0.03 秒 then
12:    repeat
13:      タスク処理時間計測
14:      end_time ← タスク終了直後の時刻
15:    until 処理時間が平均と同等
16:    if |end_time − start_time −1.5 秒 | < ϵ then
17:      { 処理時間の遅延発生から終了までの時間が 1.5 秒付近 }
18:      同期成功
19:    else
20:      同期失敗 { 負荷の原因は sender プロセス以外と判断 }
21:    end if
22:  end if
23: until 同期が成功する
24: { 受信要求受理 次の処理へ }
```

れ表現している．若干の誤差の発生が考えられるが，通信を行うにあたり特に問題にならない程度である．

スリープの後，通信開始処理の最後に，プロトコルは sender プロセスに対して 1 秒間 CPU 負荷をかけないことを要求する．よって receiver プロセスは，この 1 秒間に CPU 負荷が検出された場合，先ほど CPU 負荷をかけていたドメインは，送信側ドメインの sender プロセスではなかったと判断し，通信開始処理の最初へと戻り，sender プロセスからの受信要求を待ちなおす．このように sender プロセスに複雑な CPU 負荷のパターンを要求することで，sender プロセスが受信要求を出しているかどうかを，高い精度で識別できるようになる．またこのとき，後の通信処理のために，receiver プロセスはこの 1 秒間でタスクを何回処理できたかを基準処理回数として記録しておく．

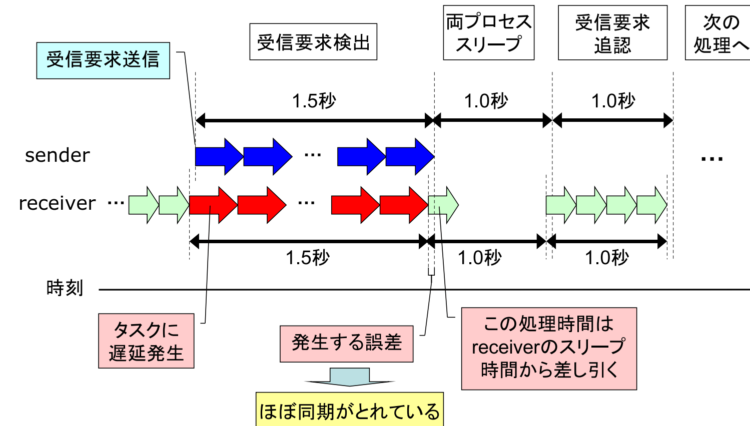

図 4 受信要求による同期

3.2.2 ビット列通信処理

通信開始処理に引き続き，sender，receiver 両プロセスは，10 進数 16 桁を表現する 64 ビットの通信処理を行う．1 ビットの基本的な通信方法は 3.1.2 節にて述べた．1 ビット分の通信には，1 秒間の CPU 負荷状況を用いる．receiver プロセスは，この 1 秒間で処理できるタスクの回数を測定し，処理できた回数が通信開始処理の最後に用意した基準処理回数の 90% 以上であれば，そのビットは 0 と解釈し，90% 未満であれば，そのビットは 1 と解釈する．すなわち，遅延があまり大きくなくても，receiver プロセスは sender プロセスが CPU 負荷をかけたと推定している．これは，受信側ドメインが送信側ドメインより優先度が高いとき (receiver プロセスに遅延が発生しにくい状態) でも，ある程度の通信が行えることを意図した．この推定方式には，通信中に発生した第三者のドメインやプロセスに引き起こされた遅延より，ビット 0 をビット 1 と誤る判断を下してしまう恐れが大きくなるという欠点がある．

© 2009 Information Processing Society of Japan

ビット列を通信するに当たって，各ビットごとに同期処理を行う必要はない．プロトコルが，sender，receiver 両プロセスに 1 ビットの送受信処理は 1 秒間という決まった時間しか行わないことを要求しているので，ビット列通信中に両プロセスの送受信処理のタイミングにズレが生じることは考えにくいからである．

またプロトコルは，通信される各ビット間 (あるビットを通信後，次のビットの通信を行う前) において，両プロセスが 1 秒間 CPU 負荷をかけないことを要求する．このようにインターバルを設けないと，receiver プロセスは常に CPU を利用している状況となり，受信側ドメインはオーバースケジュール状態となる．このとき，通信と関係のない第三のドメインが，あまり CPU 負荷をかけないような小さな処理を実行した場合でも，そちらの処理がスケジューラによって大幅に優先された結果，receiver プロセスの処理回数が低下し，ビット判定を誤る可能性が生じる．

ビット列の通信が終了したら，sender，receiver 両プロセスは，特別な処理をせずにそのまま通信を終了する．

この通信に必要となる時間は理論上，通信開始処理 3.5 秒，通信開始処理とビット列通信処理の間に発生するインターバル時間 1 秒，ビット通信に必要な時間計 64 秒，ビット間に必要なインターバル時間計 63 秒の合計 131.5 秒である．1 ビット当たりの通信は 2.0 秒で行われる．

4. 評　　価

4.1 実験環境

実験は，CPU が Celeron 2.93GHz，メモリが 1G バイトの物理マシン上で行った．Xen ハイパーバイザのバージョンは 3.1.4 であり，すべてのゲスト OS は Debian etch とし，各 domU へのメモリ配分は 128M バイトずつとした．

4.2 通信精度

ドメインの優先度や第三者のプログラムに関する状況を様々に変化させて，本システムの通信精度の測定を行った．ある 1 つの状況での試行回数は 100 回とし，それぞれ正しく同期をとって通信を開始できたか，通信されるべき 10 進数 16 桁の数字列は正しいものが通信できたか調査した．

4.2.1 他の CPU 負荷がない状態での精度

通信を行う sender プロセスと receiver プロセス以外のプロセスが，CPU をほとんど使用しない理想的な状態での精度を測定した．ただし，資源管理などのため，dom0 が CPU を 4% 程度利用することは避けられない．

表 1 他に CPU 負荷のない状態でのドメイン優先度と精度の関係

	S : R	3:1	2.5:1	2:1	1.5:1	1:1	1:1.5	1:2	1:2.5	1:3
通信結果 (回)	通信成功	92	98	100	98	100	61	54	16	4
	同期失敗	8	2	0	2	0	39	46	84	96
	誤り発生	0	0	0	0	0	0	0	0	0

このとき，送信側ドメインと受信側ドメインの優先度を変化させ，それぞれの場合で 100 回試行を行った．その結果を**表 1** に示す．この表において,「S:R」は「送信側ドメインの優先度 : 受信側ドメインの優先度」を表す.「通信成功」とは問題なく正しい情報をやりとりできた通信を,「同期失敗」とは通信開始処理において receiver プロセスが sender プロセスの受信要求を正しく受け取れなかった通信を,「誤り発生」とは通信は開始できたが，受信した情報に誤りが発生した通信をそれぞれ表す.

表 1 より，両ドメインの優先度が 1:1 という極めて理想的な環境においては，極めて高い精度で通信できるが，各ドメインの優先度の差が大きくなると，精度が低下することがわかる．特に，受信側ドメインの優先度が大きいときに顕著である．また，通信すべき情報の誤りが見られなかったことから，一方の優先度が他方の 3 倍以内であり，dom0 以外の第三者の CPU 利用がない場合，同期をうまくとることができれば，その後の通信内容についての信頼性は高いと言える．しかしながら，同期の失敗が多く見られる．これは，送信側ドメインの方が優先度が高い場合と，受信側ドメインの方が優先度が高い場合で原因が異なると考えられる．送信側ドメインの優先度が高い場合，receiver プロセスは受信要求検出中のタスク処理に際して，より多くの時間的影響を受けることになり，各タスク処理の遅延時間は大きくなる．このときタイミングによっては，receiver プロセスは 1.5 秒よりかなり長い時間 CPU 負荷がかかっていたと判断し，sender プロセスによる CPU 負荷であると認識しないことがありうる．また，受信側ドメインの優先度が高い場合，receiver プロセスの受信要求検出用のタスクに発生する遅延時間が小さく，そもそも CPU 負荷を認識していないことが考えられる．

4.2.2 他に CPU 負荷が発生している状態での精度

両ドメインの優先度を 1:1 で固定し，通信に関係のない第三の domU にて CPU 負荷が発生している状態での精度を測定した．結果を**表 2** に示す．CPU 負荷を発生させるため，第三者となる domU を 2 つ用意し，それぞれのドメイン内で Apache httpd サーバを動作させ，その Apache に別の物理マシンからリクエストを送信した．Apache は，それぞれの

© 2009 Information Processing Society of Japan

表 2 第三者によって発生する CPU 負荷と精度の関係

	CPU 負荷 (%)	7	10	15
通信結果 (回)	通信成功	92	91	74
	同期失敗	7	9	13
	誤り発生	1	0	13

リクエストに対して静的なファイルを返す．以下に，それぞれの詳しい状況を示す．

- CPU 負荷 7%
  第三者 domU のうちの 1 つに，1 秒間に 55–60 回のリクエストを送信して負荷を発生させた．このとき，送信側ドメインと受信側ドメイン以外の第三者ドメイン (dom0 も含まれる) の実 CPU 使用率は，5–9%程度の範囲内で変動していた．
- CPU 負荷 10%
  第三者 domU の両方に，1 秒間に 55–60 回のリクエストを送信して負荷を発生させた．このとき，送信側ドメインと受信側ドメイン以外の第三者ドメイン (dom0 も含まれる) の実 CPU 使用率は，8–12%程度の範囲内で変動していた．
- CPU 負荷 15%
  第三者 domU のうちの一方には 1 秒間に 55–60 回のリクエストを，他方には 1 秒間に 110–120 回のリクエストを送信して負荷を発生させた．このとき，送信側ドメインと受信側ドメイン以外の第三者ドメイン (dom0 も含まれる) の実 CPU 使用率は，13–17%程度の範囲内で変動していた．

表 2 から，第三者ドメインの CPU 利用率が，10% 以下なら 9 割程度の精度を維持できるが，これ以上 CPU 利用率が高まると通信の精度の低下を招くことがわかった．また，同期が失敗したのは，以下の 2 つの状況が重なって発生したためだと考えられる．

(1) Receiver プロセスが通信開始処理として行っている初期化の際，第三者ドメインが多くの CPU リソースを必要としていた．
(2) Sender プロセスが受信要求を送信している間，第三者ドメインはあまり CPU を利用していなかった．

このとき，(1) の場合において算出されたタスクの平均処理時間より，(2) の場合におけるタスクの処理時間が短いか同等になることがある．つまり，sender プロセスが発生させた遅延が埋没し，receiver プロセスが sender プロセスの受信要求を検出できなかったとき，同期が失敗するのだと考えられる．また，通信する情報に誤りが生じる原因としては，情報 (ビット列) の通信中に第三者ドメインによる CPU 負荷の変動が大きく，receiver プロセスがそれを sender プロセスによるものと誤認識しているからだと考えられる．

4.3 脅威レベルの考察

本研究によって，送信側ドメインと受信側ドメインの優先度が同じで，かつ第三者のドメインの CPU 利用がない状態では，1 ビット当たり 2.0 秒の時間をかけることによって，極めて高い精度で covert channel 通信が行えることがわかった．クレジットカード番号や，root のパスワードなど，比較的小さな情報量かつ極めて重要である情報が流出する通信路として，かなりの脅威となりうる．

ところが，送信側ドメインと受信側ドメインの優先度が異なる場合，通信そのものを検出できない不確実性が発生する．特に，受信側ドメインが優先されている場合に顕著になる．しかし現実的に，攻撃者が送信側ドメインとなる攻撃対象ドメインから情報を盗み出そうと考える場合，自分の管理下である受信側ドメインを，送信側ドメインより高い優先度に設定しなければならない理由はない．VPS でホスティングサービスを行っている企業 (マシン管理者) が各ドメインに優先度の設定を行っている場合，ドメイン毎の優先度はサーバの利用料金額の大小などで決定されると考えられるが，どのような方法で決定されるのであれ，優先度が低く設定される条件は，優先度が高く設定される条件を満たすより寛容であると考えられる．利用料金の例で言えば，攻撃者はサーバの利用料金として最小額を支払っておけばよい．よって攻撃者は，精度の著しい低下を防ぐため，受信側ドメインの優先度を送信側ドメインのそれと同じか，それ以下に設定されるように誘導すると考えられる．またマシン管理者は，商用のサービスを提供するにあたって，それぞれのドメイン間に 3 倍を超える優先度差を設けるとは考えにくい．送信側ドメインの優先度が 3 倍であっても 9 割以上の通信が行えたことから，同じ通信を複数回行うことによって，かなり高い精度で通信を行うことができるようになる．ゆえに，両ドメインの優先度が異なる場合であっても，脅威レベルは高いと言える．

また，他のドメインで CPU 負荷が発生している場合，その処理量の変動から，たびたび同期失敗や通信に誤りが発生しているが，おおむね 10%以下であれば 9 割以上の精度を維持しているので，やはり複数回の通信を行うことによって，より信頼性の高い通信を行うことができるようになると考えられる．130 秒強の通信中に負荷が大きく変動しないことが条件となるが，早朝などの処理を行っているドメインが少ないと考えられる時間帯に使用することで，脅威レベルは決して低いものではなくなるだろう．

© 2009 Information Processing Society of Japan

5. 関 連 研 究

CPU を用いた仮想マシン間の covert timing channel として，SMT プロセッサ内回路の演算器を用いたものがある[4]．この covert channel 通信は，乗算器などの演算器の使用時間を送信側プロセスが調節し，受信側プロセスはその演算器の利用にかかった時間によりビット判定を行うものである．演算器を利用することで，よりタイトなスケジューリングを行うことができ，通信速度は高速となるが，当然ながら CPU として SMT プロセッサを用いていなければ通信を行うことはできない．

他の類似の covert channel としては，バス競合などのハードウェア上で発生しうる遅延を用いた timing channel がある[5] が，VAX アーキテクチャ上で行われた研究であり，現代的な仮想マシンモニタ (ハイパーバイザ) 上の covert channel を扱ったものではない．

Matthews らの研究[6] において，Xen の CPU に関するパフォーマンスの隔離は良好で，ある 1 つの仮想マシンが 100%近い CPU 負荷を発生させても，同仮想マシン内で動作する web サーバや，他の仮想マシン上で動く web サーバの性能低下はほとんどなかったことが示されている．つまりこの研究からは，プロセススケジューラやドメインスケジューラが優秀であることがわかる．しかし一方で，さらに別の仮想マシンにおいても 100%近い CPU 負荷を求められた場合において，それぞれの仮想マシン内の高い CPU 負荷を要求しているプロセスの性能低下が発生しないことを示したものではなく，我々の主張とは趣を異にする．我々の研究は，この性能低下を捉えて通信を行っている．

sHype[7] は，Xen ハイパーバイザに適用できる強制アクセス制御機構である．covert channel 通信を防ぐ意味でも有用であり[8]，特に Chinese Wall ポリシーを適用されると，本研究の covert timing channel 通信が阻害される可能性がある．このポリシーは，ある仮想マシンと同じ集合内に存在する仮想マシンは，そのある仮想マシンと同時に動作しないことを保証するものである．しかしながらこのポリシーを適用するには，あらかじめ管理者が covert channel 通信が行われる危険性がある仮想マシン同士を認識していなければならず，有効な運用は難しい．またこのポリシーを適用すると，同時に動作できる仮想マシンが制限され，性能低下を招く恐れがある．

6. まとめと今後の課題

本研究では，covert channel として実 CPU を用いることで，Xen 上のドメイン間をまたがる通信を行うシステム CCCV を構築し，この通信の評価を行った．構築した CCCV はゲスト OS のユーザプロセスとして動作するので，カーネル権限を必要としない．

評価の結果，ある程度の前提条件が必要にはなるが，1 ビットあたりの通信時間が 2.0 秒でかなり高い精度を保っており，比較的小さな情報量かつ極めて重要である情報が流出する通信路として，かなりの脅威となりうることを確認した．特に，環境や条件によってはほぼ 100% の通信が行えることを示せ，対策の必要性を感じさせる結果が得られた．

今後の課題として，時間をおいて複数回の通信を行ったり，誤り訂正符号を用いることによる通信のさらなる精度向上を目指すことや，反対に，CPU 負荷をランダムに発生させるなどして，この通信の妨害を行う防御手法を考案することが挙げられる．また，マルチコア環境での実験を行う予定である．

参 考 文 献

1) Lampson, B.W.: A Note on the Confinement Problem, *Communications of the ACM*, Vol.16, No.10, pp.613–615 (1973).
2) Center, N. C.S.: *A Guide to Understanding Covert Channel Analysis of Trusted Systems*, NCSC-TG-30 (1993).
3) 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, *Proceedings of the 19th ACM SOSP*, pp.164–177 (2003).
4) Wang, Z. and Lee, R.B.: Covert and Side Channels Due to Processor Architecture, *Proceedings of the 22nd Annual Computer Security Applications Conference*, pp.473–482 (2006).
5) Hu, W.-M.: Reducing timing channels with fuzzy time, *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, pp.8–20 (1991).
6) Matthews, J.N., Hu, W., Hapuarachchi, M., Deshane, T., Dimatos, D., Hamilton, G., McCabe, M. and Owens, J.: Quantifying the Performance Isolation Properties of Virtualization Systems, *Proceedings of the 2007 Workshop on Experimental Computer Science* (2007).
7) Sailer, R., Jaeger, T., Valdez, E., Cáceres, R., Perez, R., Berger, S., Griffin, J.L. and van Doorn, L.: Building a MAC-Based Security Architecture for the Xen Open-Source Hypervisor, *Proceedings of the 21st Annual Computer Security Applications Conference*, pp.276–285 (2005).
8) Trent, J., Reiner, S. and Yogesh, S.: Managing the Risk of Covert Information Flows in Virtual Machine Systems, *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, pp.81–90 (2007).

© 2009 Information Processing Society of Japan