仮想マシンモニタを用いたマルウェア実行抑止機構

岡 村 圭 祐† 大 山 恵 弘†

1. はじめに

近年，ホスティングサービスの一つの形態として，事業者が仮想マシン単位で計算機資源をユーザ（サービスの受け手）に貸し出し，仮想マシン上で動作するゲスト OS の root 権限もユーザに与える，“IaaS” が注目されている．IaaS を提供している事業者の例として，Amazon[1] などが挙げられる．

IaaS においてユーザは，仮想マシン（VM）と呼ばれる仮想的な計算機資源を管理する．この VM 内の OS を,「ゲスト OS」と呼ぶ．VM は，仮想マシンモニタ（VMM）を用いて実現される．このとき，VMM 上で動く各 VM は，その VM を割り当てられた各ユーザによって管理され，VMM や物理マシンは IaaS プロバイダによって管理される．つまり，VMM の管理者と各 VM の管理者は異なる．

さてここで，ある VM 上のゲスト OS がマルウェアに感染している事態を考える．このマルウェアが有害な処理を行っていたり，無意味なループの実行などにより，計算機資源を不当に占有している場合，他のユーザの VM への計算機資源割り当て量が減少し，処理速度の低下を招く恐れがある．そこで VMM の管理者（IaaS プロバイダ）は，サービスの品質低下を嫌い，このマルウェアへの素早い対処を望むだろう．しかしながら現状では，VMM 層からゲスト OS 内のマルウェアの検知はできる[3][4][5] ものの，そのマルウェアに対処することは困難である．すなわち，IaaS プロバイダがゲスト OS 上でマルウェアが動作していることを発見しても，そのマルウェアを IaaS プロバイダ自身が駆除することは難しい．これは，IaaS のような利用場面では VMM とゲスト OS の管理者が異なっており，IaaS プロバイダはゲスト OS のデータを勝手に書き換えるわけにはいかないからである．

2. 目的と方針

本研究では，ゲスト OS 内で動作しているマルウェアに対し，VMM 層から対処する機構を提案する．本機構は，VM 内で動作しうる OS の種類の多くに対応するため，ゲスト OS の実装に依存しない手法をとる．

提案する手法は，仮想タイマ割り込みのタイミングの変更と，システム時間のエミュレートからなる．これにより，任意のプロセスの実行速度を大幅に低下させる．つまり，マルウェアの動作を妨害できるようになる．なお，マルウェアを kill するには，ゲスト OS の知識（実装）が必要になるため，本機構ではマルウェアを完全に無効化せず，あくまで実行速度の低下を強制する．我々は，IaaS プロバイダがマルウェアの実行を抑止した上で，マルウェアが動作している VM のユーザに，マルウェアへの対応を求めることを想定している．つまり本機構は，マルウェア発生時に初期対応を行うための機構と位置付ける．本機構は Xen[2] の改変にて実現し，ゲスト OS のカーネルを変更しない．

3. タイマ割り込みとシステム時間

OS のプロセススケジューラは通常，タイマ割り込みを契機に，現在動作中のプロセスがその割り当てられた CPU 時間（タイムスライス）を使い切ったか否かを判定し，使い切っているのであれば，別のプロセスをスケジューリングする．現代的な OS において，この判定にはタイマ割り込みの回数や，システム時間が用いられると考えられる．システム時間とは，OS がブートしてからの経過時間である．

実機（物理マシン）上で直接動作している OS は当然，タイマ割り込みは一定間隔で発生し，ハードウェアから取得するシステム時間は正しいものであると信じている．これは，Xen 上のゲスト OS であっても同じである．それゆえ Xen には，一定間隔で仮想タイマ割り込みを発行し，正しいシステム時間をゲスト OS に提供する責任がある．

4. 提案手法

提案手法では，タイマ割り込みの間隔を短くし，システム時間の経過速度を速くすることで，マルウェアの実行速度を低下させる．以下で例を用いて説明する．

† 電気通信大学

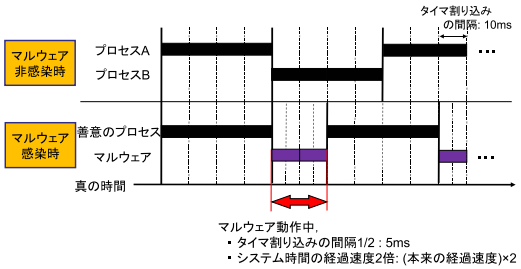

図 1 タイマ間隔の短縮と時間の経過延長に伴うスケジューリングの変化

今，ドメイン内でプロセス A とプロセス B の 2 つのプロセスが動作しているとし，プロセススケジューラは，40ms ごとにこれらを交互にスケジューリングすると仮定する．このとき，これらのプロセスが割り当てられる VCPU は 1 つであるとする．仮想タイマ割り込みは通常，10ms ごとに発行される．この割り込みを契機にプロセススケジューラは，そのプロセスがスケジューリングされてから 40ms が経過したか判定する．この判定のための時間は，システム時間から算出される．このときの CPU 利用状況を，**図 1** の上段に示す．図中黒塗りの部分が，CPU を利用していることを表す．また破線はタイマ割り込みのタイミングを表し，実線はプロセスが切り替わるタイミングを示す．

さてここで，一方のプロセスがマルウェアであり，もう一方のプロセスは善意のプロセスである場合を考える．このとき VMM は，マルウェアの実行中のみ仮想タイマ割り込みの間隔を，例えば 1/2 である 5ms とする．さらにこの割り込みの配送時に，実際に経過した時間である 5ms の 2 倍，すなわちあたかも 10ms 経過したかのように書き換える（図 1 下段）．つまり OS にとって，マルウェアの動作中は 2 倍の速さで時間が経過することになる．この結果，OS はマルウェアの CPU 使用時間を誤認識し，本来与えるべき時間である 40ms の半分，20ms しか与えずにプロセス切り替えを行う．このとき OS は，マルウェアに十分な時間を与えていると誤解している．これらは，ゲスト OS の外からマルウェアの実行速度を 1/2 にすることができることを示している．

5. 予 備 実 験

提案手法によって特定のプロセスの実行速度を変更できるかを，実験により確認した．実験は，CPU が Intel Core 2 Duo 2.53GHz，メモリが 2G 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 4.0.0 とした．提案する手法を適用するためのゲスト OS として Debian lenny (Linux 2.6.26) を準仮想化し用いた．また，このゲスト OS には，メモリを 512M バイト割り当てた．

対象となるゲスト OS 内で，CPU を集中的に使用しながら一定の計算を繰り返すプロセスを複数動作させ，そのうちの 1 つをマルウェアと見立てた．この「マルウェア」が動作している間のみ，タイマ割り込みとシステム時間を操作し，割り込みの間隔を 1/1000 に，システム時間の経過を 1000 倍にした．これらのプロセスには，一定の計算を終えるごとにその旨を標準出力に出力させた．このそれぞれの出力の頻度を比較することで，各プロセスの速度を見積もった．

対象となるゲスト OS のもつ VCPU が割り当てられる PCPU を，ある 1 つに固定した場合，VCPU の個数にかかわらず，特定プロセスの実行速度を低下させることに成功し，およそ 1/4000 の速度に減じることができた．この結果より，提案手法の実現可能性は充分に高いと考えられる．

6. 現状と今後

仮想タイマ割り込みの間隔の操作と，システム時間のエミュレートを行う機構を実装した．また，提案機構の実現可能性を調査するため実験を行い，対象となるプロセスの実行速度が低下したことを確認した．

今後は他の検知機構と連携させ，有効性を評価する．

謝　　辞

本研究の一部は科研費（19700024）の助成を受けている．

参 考 文 献

1) Amazon Elastic Compute Cloud (EC2). `http://aws.amazon.com/ec2/`
2) P. Barham, *et al.* Xen and the Art of Virtualization, In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 164–177, 2003.
3) X. Jiang, *et al.* Stealthy Malware Detection Through VMM-Based “Out-of-the-Box” Semantic View Reconstruction, In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 128–138, 2007
4) B.D. Payne, *et al.* Lares: An Architecture for Secure Active Monitoring Using Virtualization, In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, pages 233–247, 2008
5) S.T. Jones, *et al.* Antfarm: Tracking Processes in a Virtual Machine Environment. In *Proceedings of the USENIX Annual Technical Conference*, pages 1–14, 2006.