Vol.2011-OS-116 No.3
2011/1/24

踏み台攻撃だけを抑制できる VMM レベル・パケットフィルタ

安積武志[†1] 光来健一[†2,†3] 千葉滋[†1]

クラウドコンピューティングにおいて、ユーザに提供している仮想マシン（VM）からの踏み台攻撃はデータセンタにとって大きな問題である。VM から踏み台攻撃が行われると、データセンタが攻撃者とみなされる可能性がある。ファイアウォールで踏み台攻撃の通信を遮断することができるが、踏み台にされた VM からの通信を完全に遮断してしまうとサービス可用性が低下してしまう。高いサービス可用性を保つため、我々は仮想マシンモニタ（VMM）で動作するきめ細かいパケットフィルタ xFilter を提案する。xFilter は VM のメモリ解析を行って取得したゲスト OS 内の情報を用いることで、踏み台攻撃を行っているプロセスからのパケットのみを破棄する。踏み台攻撃を検出する侵入検知システムも VMM で動作させることで、パケットの送信元を特定する精度を高めている。いくつかの最適化を行うことで xFilter のオーバーヘッドを小さくすることができた。

A VMM-level Packet Filter Preventing Only Stepping-stone attacks

Takeshi Azumi,[†1] Kenichi Kourai[†2,†3] and Shigeru Chiba[†1]

In the cloud computing era, stepping-stone attacks via hosted virtual machines (VMs) are critical for data centers. When VMs attack external hosts, data centers may be regarded as attackers. External firewalls are useful for stopping such attacks, but the service availability of stepping-stone VMs remarkably lowers if all packets from the VMs are dropped. For higher service availability, we propose a fine-grained packet filter running in the virtual machine monitor (VMM), which is called xFilter. xFilter drops only packets from processes performing stepping-stone attacks by using information in guest operating systems. It analyzes the memory of VMs to obtain such information. An intrusion detection system in the VMM accurately specifies attacking processes. Our experimental results show that xFilter achieves low overheads thanks to several optimizations.

1. はじめに

クラウドコンピューティングを提供しているデータセンタにとって**踏み台攻撃** は大きな脅威となっている。踏み台攻撃では、攻撃者自身のホストからではなく前もって侵入したホストから攻撃を行う。データセンタ内のホストが外部ホストへの攻撃の踏み台として利用された場合、侵入されたホストのユーザだけでなくデータセンタも攻撃の責任を負うことになる可能性がある。データセンタは被害者であると同時に、攻撃者にもなってしまう。特に、Amazon EC2 のような Infrastructure as a Service（IaaS）と呼ばれるサービスモデルは踏み台攻撃に対してより脆弱である。IaaS はユーザに仮想マシン（VM）を提供し、ユーザは OS とその上のソフトウェアをインストールする。データセンタにとってすべての VM 内のすべてのソフトウェアに対して、すべてのセキュリティパッチが適用されていることを保証することは困難である。

このように、完全にユーザの VM への侵入を防ぐことは困難であるため、踏み台攻撃をできるだけ早く止めることが重要になる。外部ファイアウォールによるパケットフィルタリングは最も安全で確実な方法である。ある VM からの踏み台攻撃が検出されたら、外部ファイアウォールは VM からの全パケットを拒否すればよい。この単純なフィルタリングルールによって、侵入された VM からの踏み台攻撃を完全に遮断することができる。しかし、踏み台にされた VM 内のサービスの可用性は著しく低下してしまう。この VM 内の正常なアプリケーションも外部にパケットを送信できなくなるためである。ファイアウォールで攻撃に利用されている特定のポートへのパケットのみを拒否したとしても、正常なアプリケーションはそのポートを使ってパケットを送信できなくなる。

本稿では、仮想マシンモニタ（VMM）内におけるきめ細かいパケットフィルタである *xFilter* を提案する。安全性に関しては、xFilter は VM から隔離されているため、xFilter 自身が VM 内の侵入者から攻撃される可能性は低い。サービス可用性に関しては、xFilter はゲスト OS 内の情報を利用することで踏み台攻撃だけを防ぐことができる。本来、VMM

†1 東京工業大学
Tokyo Institute of Tecnology
†2 九州工業大学
Kyushu Institute of Technology
†3 独立行政法人科学技術振興機構, CREST
Japan Science and Technology Agency, CREST

は VM 内のゲスト OS の内部構造を理解できないが、xFilter は VM のメモリを解析して必要な情報を取得する。例えば、xFilter は攻撃パケットの送信元を指定することによって、特定のプロセスから送信されたパケットのみを拒否することができる。また、侵入検知システム（IDS）も VMM 内で動作させることで、踏み台攻撃を検出するとすぐに送信元を特定する。このように、パケットフィルタを VMM 内で動作させるとそのバグがシステム全体に影響を与えやすくなるため、xFilter はそのモジュールの開発サポートを提供している。開発者はモジュールを別の VM 上でテストし、変更なしに VMM 内に埋め込むことができる。

我々は xFilter を Xen に実装した。xFilter はパケット送信処理の途中で VM のメモリ解析を行うため、その性能がネットワーク性能に大きな影響を及ぼす。メモリ解析のオーバーヘッドを削減するために、我々はいくつかの最適化を行った。ゲスト OS 内の情報を利用する多くのシステム[8),9)] と違い、xFilter はドメイン 0 と呼ばれる特権 VM ではなく、VMM 内でメモリ解析を行う。VMM からはオーバーヘッドなしで直接 VM のメモリにアクセスすることができる。さらに、同一 TCP コネクション内のパケットのフィルタリング結果をキャッシュすることによってメモリ解析の回数を減らす。これらの最適化によって、xFilter は標準的なプロセス数、ソケット数の場合にネットワーク性能の低下を 4%以下に抑えることができた。

以下、2 章では踏み台攻撃に対する既存の対処法について述べる。3 章では VMM で動作する新しいパケットフィルタ、xFilter を提案し、4 章では Xen における実装について説明する。5 章では xFilter のオーバーヘッドを測定した実験の結果を示す。6 章で関連研究に触れ、7 章で本稿をまとめる。

2. 踏み台攻撃への対処法

踏み台攻撃は侵入された VM のユーザだけでなく、その VM を提供しているデータセンタの信用にも関わるため、できるだけ早く止める必要がある。データセンター内の VM が踏み台として利用されないようにすることが望ましいが、ソフトウェアには多くの脆弱性があるためそれは難しい。特にクラウドコンピューティングにおいては、物理ホストと違って必要な時だけ VM を動作させる。長い間動作させていなかった VM にはセキュリティパッチが適用されていないため、攻撃を受けやすい。また、セキュリティはパスワード管理などのようにユーザの教育にも依存する。幸い、踏み台として使われているコネクションの検知[2),4),13),15)] や踏み台ホストからのポートスキャンの検知[12)] などの様々な検知技術が提案されている。

データセンタにとって、外部ファイアウォールによるパケットフィルタリングが踏み台攻撃への最も簡単で確実な対処法である。外部ファイアウォールはデータセンタのネットワークの出入り口のように、踏み台にされる VM の外部に置かれている。VM への侵入者が外部ファイアウォールを攻撃することは困難なので、外部ファイアウォールによるパケットフィルタリングは安全である。しかし、外部ファイアウォールは IP アドレスやポート番号といった、パケットに含まれる情報しか検査できないため、踏み台 VM のサービス可用性を保つのが難しい。例えば、送信元の IP アドレスに基づくパケットフィルタリングが最も容易に適用できる。踏み台 VM からのパケットすべてを拒否するルールを追加すれば、踏み台攻撃を完全に止めることができる。その一方で、VM 内の正常なアプリケーションも外部にパケットを送信できなくなってしまう。その結果、踏み台 VM のサービス可用性はゼロになる。

パケットの他の情報を利用することでサービス可用性を向上させることもできる。送信先の IP アドレスに基づくパケットフィルタリングを行えば、特定のホストへのパケット送信だけを遮断することができる。このようなフィルタリングは特定ホストへのサービス妨害攻撃のように、攻撃対象ホストが少ないときは有効だが、対象ホストが多い場合にはすべてのホストを指定するのは難しい。侵入者が多くのホストに対して SMTP スキャンを行っているような場合は、送信先のポート番号に基づくパケットフィルタリングを行えば、すべてのホストの特定サービスへのパケット送信のみを遮断できる。侵入者はどのホストに対しても SMTP スキャンを行えなくなるが、正常なアプリケーションもメールを送れなくなる。一方、送信元のポート番号を指定すれば、特定のコネクションだけを制限することができる。SMTP スキャンのような短時間のコネクションには効果がない。

他方、VM 内部のファイアウォールを利用すれば、高いサービス可用性を実現できる。このファイアウォールは OS カーネル内にあるため、パケットフィルタリングに送信元の情報も利用できる。例えば、Linux の iptables や FreeBSD の ipfw は、フィルタリングルールにプロセス ID やユーザ ID を指定できる。侵入者の使用しているプロセスを特定することで、侵入者が ssh を行ったときのみパケットを拒否することができ、それ以外の正常なアプリケーションは ssh を行うことができる。しかし内部ファイアウォールは、踏み台攻撃に対して脆弱である。侵入者に管理者権限を奪われると、踏み台攻撃を防ぐためのフィルタリングルールを削除することで、内部ファイアウォールを簡単に無効化される。さらに、データセンタの管理者は VM にログインする権限も、内部ファイアウォールにルールを追加する

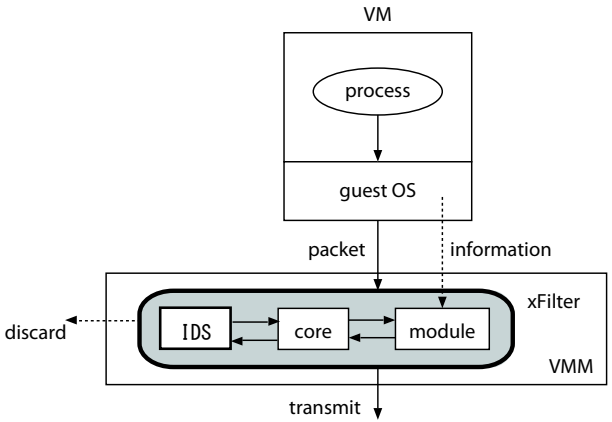


図 1 VMM 内で動作する xFilter

権限もないことが多い。そのため、データセンタの管理者が攻撃に気づいたとしても、VM のユーザに通知して踏み台攻撃を止めるには時間がかかってしまう。

本稿で扱う踏み台攻撃では、攻撃者に VM に侵入され、ゲスト OS の管理者権限を奪われることがあると仮定している。ただし、ゲスト OS のカーネルは改ざんされていないものとする。OS カーネルの改ざんは VMM が検出することができる[3),7)]。また、セキュリティハードウェアが VMM の整合性を保証できる[10),14)] ため、VMM も改ざんされていないものとする。

3. xFilter

安全性とサービス可用性を両立させる踏み台攻撃への対処法として、我々は新しいパケットフィルタ xFilter を提案する。

3.1 VMM におけるきめ細かいフィルタリング

xFilter は図 1 のように VMM 内で動作するパケットフィルタである。VMM は全ての VM から隔離されているため、VM への侵入者が VMM 内で動作する xFilter を攻撃することは難しい。さらに、全てのネットワークパケットは VMM を介して外部に送られるため、xFilter は VM からの全てのパケットを検査できる。外部ファイアウォールと違い、同一ホスト内の VM 間のパケットを遮断することで、同一ホスト内のある VM から別の VM への踏み台攻撃を防ぐこともできる。

フィルタリング対象の VM のサービス可用性を向上させるため、xFilter は VM のメモリを解析することでゲスト OS 内の情報を利用する。これにより、内部ファイアウォールで利用できるものと同様の情報を取得できる。例えば、xFilter はフィルタリングルールとして、送信元のプロセスや所有者の ID を指定することができる。このようにきめ細かいルールによって、踏み台攻撃を行っているプロセスやユーザからのパケットのみを遮断することができる。従来、VMM はゲスト OS の内部構造を理解しないため、ゲスト OS の内部の情報を参照することはできなかった。その意味では、VMM は踏み台 VM から完全に独立している外部ファイアウォールに似ている。しかし VMM は VM のメモリにアクセスできるという点で、外部ファイアウォールとは異なっている。xFilter はゲスト OS に関する知識を利用することで、そのデータ構造を理解する。

xFilter ではパケットフィルタリングの実装をモジュールとして分離している。これは様々なゲスト OS に柔軟に対応できるようにするためである。ゲスト OS ごとに異なるメモリ解析が必要になる。xFilter モジュールは VMM に到着したパケットの情報を xFilter 本体から受け取り、フィルタリングルールに基づいてそのパケットを許可するか破棄するかを決定する。その際にゲスト OS のメモリ解析を行い、フィルタリングに必要な情報を取得する。例えば、特定のプロセスが送信したパケットかどうか調べるには、ゲスト OS のメモリからそのプロセスを探し出し、そのプロセスが行っている通信の一覧を取得して、受け取ったパケットの情報と比較する。

3.2 踏み台攻撃を遮断するルールの生成

VMM 内で動作する IDS が踏み台攻撃を検出すると、xFilter は攻撃のために送信されるパケットだけを破棄するフィルタリングルールを追加する。この際にモジュールを呼び出し、送信元のプロセスの情報など、ゲスト OS 内の情報もルールに付加する。従来の IDS を外部で動かすと、攻撃の検知から送信元プロセスの特定までに大きなタイムラグが発生してしまう。そのため、送信元プロセスが終了してしまっていたり、ソケットをクローズしてしまっていて、送信元プロセスが特定できない可能性がある。VMM 内に IDS を組み込み、パケット送信時に攻撃の検出を行うことで、送信元の情報をより確実に取得することができる。

攻撃パケットを送信するプロセスが刻々と変化すると、ルール数が増加し続けるため、xFilter は複数のルールの共通部分を抽出して自動でルールの最適化を行う。例えば、送信元プロセスは次々に変わっていてもそれらのプロセスの所有者が同一であれば、プロセス ID を指定したルールをそのユーザの ID を指定したルールで置き換える。このような最適

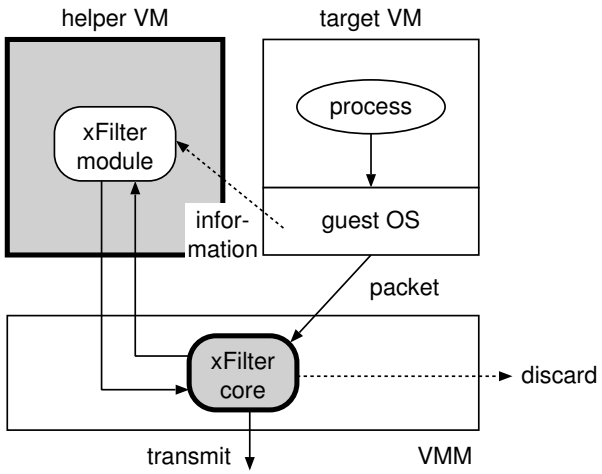


図 2 開発時の xFilter の構成

化を行うことで、ルール数を減らしながら、以降の同様の攻撃を遮断できる。プロセス ID の代わりにユーザ ID を指定することで、そのユーザが実行している正常なプロセスからもパケットを送信することができなくなってしまうが、踏み台攻撃を効果的に防ぎつつ、可能な限りサービス可用性を保つことができる。

3.3 モジュールの開発サポート

xFilter モジュールは VMM の一部であるため、その開発は容易ではない。開発者が新しいモジュールを実装したり、既存のモジュールを拡張する場合、VMM を修正する必要があるが、これには 2 つの問題がある。1 つ目は、モジュールに少しでも修正を加えたら、それを有効にするためにシステム全体を再起動する必要があることである。開発時は、モジュールへの変更が頻繁に行われる。2 つ目は、モジュールにバグがあった場合、VMM がクラッシュしてシステム全体の再起動が必要になることである。VM のメモリ解析を行うプログラムは 4 章で示すようにカーネルのデータ構造を VM の外部から間接的に扱わなければならないため複雑であり、開発の初期段階では多くのバグがある。システム全体の頻繁な再起動は開発効率を低下させる。

xFilter にはデバッグのために、図 2 のように、**ヘルパー VM** と呼ばれる別の VM 上でモジュールを動作させることができる。モジュールをヘルパー VM 上のプロセスとして動作させることによって、プロセスを実行しなおすだけで新しいモジュールを有効にできる。モジュールがクラッシュした時、開発者はプロセスを再実行するだけでよい。モジュールのクラッシュは、他のプロセスやヘルパー VM のゲスト OS、VMM、対象 VM に影響を与えない。ヘルパー VM 内のモジュールは VMM 内の xFilter 本体から呼び出され、VMM の機能を利用して対象 VM のメモリを解析する。

新しいモジュールの開発が完了したら、開発者はモジュールに修正を加えることなく VMM に埋め込むことができる。xFilter は VMM 内のモジュールとヘルパー VM 内のモジュールに同じ API を提供しているためである。モジュールが動作している場所による違いは xFilter の提供する API によって隠蔽される。例えば、VMM とヘルパー VM で他の VM のメモリにアクセスする方法は異なるが、xFilter は同じ関数を提供している。モジュールを VMM に埋め込みむことは、性能の点から必須である。モジュールをヘルパー VM 上で動作させると、パケットフィルタリングの性能が大幅に低下する。この場合、xFilter 本体は直接モジュールを呼び出すことができないため、ヘルパー VM とモジュールプロセスがスケジューリングされるのを待つ必要がある。また、ヘルパー VM から対象 VM のメモリへのアクセスには時間がかかる。

4. 実　　装

我々は、xFilter を Xen 3.4.2 に実装した。Xen は**ドメイン 0** と呼ばれる特権 VM と**ドメイン U** と呼ばれるユーザ VM を提供しており、ドメイン 0 はドメイン U の I/O 処理を行う。ドメイン U 上で動作するゲスト OS として、x86_64 向け Linux 2.6.18 を対象にした。

4.1 システム構成

図 3 に Xen における xFilter のシステム構成を示す。ドメイン U 内で `send` システムコールが発行されると、ドメイン U 内の OS カーネルは、*netfront* と呼ばれるデバイスドライバにパケットを送る。netfront はドメイン 0 のカーネル内の *netback* と呼ばれるデバイスドライバにパケットを渡す。netback ドライバは物理 NIC デバイスドライバを呼び出す代わりに xFilter を呼び出す。xFilter がパケットの送信を許可すれば、パケットはネットワークドライバに送られ、NIC に渡される。

運用時には、モジュールは性能のために VMM 内で動作する。VMM 内のモジュールを呼び出すために、ドメイン 0 内の xFilter 本体は VMM にハイパーコールを発行する。ハイパーコールはパケットを送信したドメイン U の ID とパケット情報を引数にとり、xFilter モジュールにリンクされたスタブを呼び出す。スタブは指定したドメイン U を停止してか

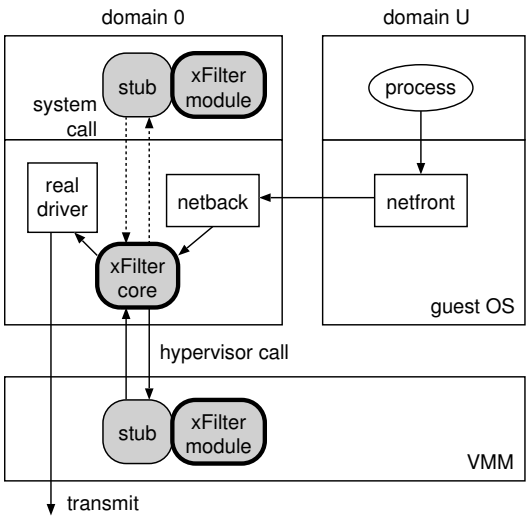


図 3 Xen 上の xFilter のシステム構成

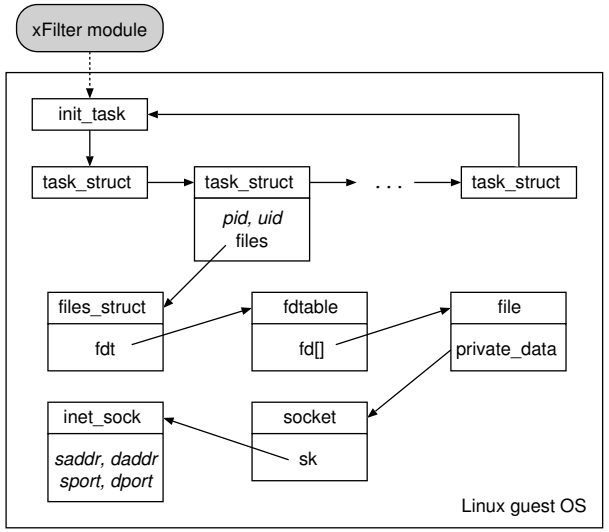


図 4 メモリ解析によるプロセス情報の取得

らモジュールを呼び出す。VMM 内のモジュールはドメイン U のメモリに直接アクセスすることでメモリ解析を行う。フィルタリング結果はハイパーコールの返り値として xFilter 本体に返される。xFilter 本体は 1 つ以上のフィルタリングルールがあるときのみモジュール呼び出すため、踏み台攻撃が検出されるまではパケットは即座にネットワークドライバに渡される。

一方、開発時には、モジュールはヘルパー VM であるドメイン 0 内のプロセスとして動作する。モジュールにリンクされたスタブはシステムコールを発行し、xFilter 本体が netback ドライバからパケットを受け取るのを待つ。パケットを受け取ったら、xFilter 本体はモジュールのプロセスを起こし、システムコールの返り値でパケットを送信したドメイン U の ID とパケット情報を返す。スタブは VMM の機能を利用してドメイン U を停止し、モジュールを呼び出す。モジュールが動作するドメイン 0 はドメイン U から隔離されているため、モジュールは VMM の機能を利用して間接的にドメイン U のメモリにアクセスする。スタブは次のパケットを待つためのシステムコールを発行することで xFilter 本体にフィルタリング結果を渡す。

4.2 ゲスト OS のメモリ解析

xFilter はゲスト OS 内のデータ構造とシンボルのアドレスを取得するために、OS カーネルのデバッグ情報を利用する。Linux の場合、プロセスの ID や所有者の情報は task_struct 構造体に格納されており、init_task シンボルが init プロセスの task_struct 構造体に対応する。xFilter はデバッグオプションつきでコンパイルしたカーネルイメージからこれらの情報を取得する。

メモリ解析の例として、Linux ゲスト OS を対象に、パケットを送信したプロセスの情報を利用するモジュールを考える。図 4 は関係するデータ構造をたどる手順を示している。まず、モジュールは init_task を起点として、ドメイン U 内のプロセスリストをたどる。プロセスリストをたどりながら、プロセスの ID または所有者がフィルタリングルールと一致するかどうかをチェックする。両方がどのルールとも一致しなかった場合、次のプロセスをチェックする。プロセス ID または所有者が少なくとも 1 つのルールに一致したプロセスに関しては、そのプロセスが使っているネットワークソケットを検査する。最終的にモジュールは、inet_sock 構造体から送信元と送信先の IP アドレスとポート番号を取得する。

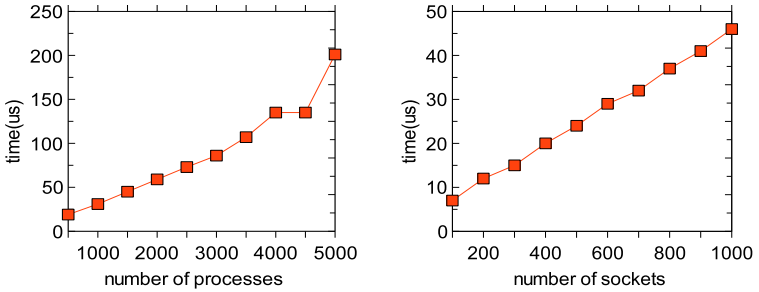
4.3 フィルタリング結果のキャッシュ

ゲスト OS のメモリ解析の回数を減らすために、xFilter はフィルタリング結果のキャッシュを行う。これにより、TCP コネクションを流れるパケットについては最初のパケットについてだけメモリ解析を行えばよくなる。xFilter がパケット送信を許可すると判断した時、フィルタリング結果をパケット情報とともにキャッシュに保存する。同じコネクションを流れるパケットはキャッシュにヒットするため、xFilter モジュールを呼び出すことなくパケット送信を許可する。xFilter がモジュールを呼び出したとしても、基本的にはキャッシュされたフィルタリング結果と同じになる。送信元プロセスの所有者が変わったり子プロセスが同じコネクションを使ったりした場合、モジュールによるフィルタリング結果とキャッシュは異なる場合がある。しかし同一コネクション内のパケットは元のプロセスや所有者と関係あると考えられるため、xFilter はキャッシュされたフィルタリング結果を適用する。

TCP コネクションに関しては、xFilter はパケットヘッダ内の TCP 制御ビットを基にキャッシュを管理する。xFilter が SYN フラグのセットされたパケットを受け取るとモジュールを呼び出し、送信を許可する場合はキャッシュにエントリを追加する。SYN フラグは新しいコネクションを確立するときにセットされる。キャッシュエントリはパケット情報として送信元と送信先の IP アドレスとポート番号を持つ。xFilter が FIN または RST フラグのセットされたパケットを受け取ったら、キャッシュから一致するエントリを削除する。FIN フラグは既存のコネクションを終了するときにセットされ、RST フラグはコネクションをリセットするときにセットされる。上記のフラグがセットされていないパケットに関しては、xFilter はキャッシュを調べ、一致するエントリがない場合だけモジュールを呼び出す。

5. 実　　験

xFilter のオーバーヘッドを調べるために、VM のメモリ解析のオーバーヘッドおよびウェブサーバの性能の低下を測定した。xFilter を動作させるサーバは、Intel Core i7 860 の CPU を 1 基、メモリ 8GB、ギガビットイーサネットを搭載した計算機を使用した。VMM としては Xen 3.4.2 、VM 内で動かす OS には Linux 2.6.18 を用いた。Xen のドメイン 0 にはメモリを 7GB、ドメイン U にはメモリを 1GB 割り当てた。ドメイン U 上で Apache ウェブサーバ 2.0 を動かした。一方クライアントには、Athlon 64 Processor 3500+ の CPU を 1 基、メモリ 2GB、ギガビットイーサネットを搭載した計算機を使用した。これらの計算機はギガビットスイッチで接続した。



(a) プロセス数を変更した場合　　(b) ソケット数を変更した場合

図 5　メモリ解析時間

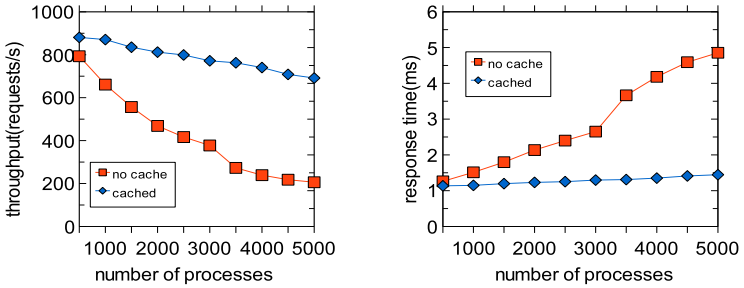
5.1 メモリ解析のオーバーヘッド

VM のメモリ解析のオーバーヘッドを調べるため、netback ドライバから呼び出す VM のメモリ解析のためのハイパーコールの実行時間を測定した。まず、プロセス ID に基づくフィルタリングルールを用いて、プロセス数を変えて実験を行った。存在しないプロセスの ID を指定することで xFilter モジュールはプロセスリストの全エントリをたどり、全プロセスのプロセス ID を調べる。指定された ID のプロセスが存在しないため、xFilter モジュールはネットワークソケットを調べない。図 5(a) はメモリ解析にかかる時間を示している。実行時間はほぼプロセス数に比例し、プロセス毎に 40ns である。5000 プロセスで 201μs になるが、一般的にはこれほど多くのプロセスが動作することはない。より現実的な 500 プロセスでは 19μs であった。

次に、ユーザ ID に基づくフィルタリングルールを用いて、プロセスがオープンしているソケット数を変えて同様の実験を行った。xFilter モジュールはプロセスリストをたどり各プロセスのユーザ ID を調べる。ユーザ ID がフィルタリングルールに指定したものであったら、xFilter モジュールはソケットまでデータ構造をたどっていく。図 5(b) はその結果を示している。実行時間はほぼソケット数に比例し、ソケット毎に 46ns である。1000 ソケットでは 46μs であった。

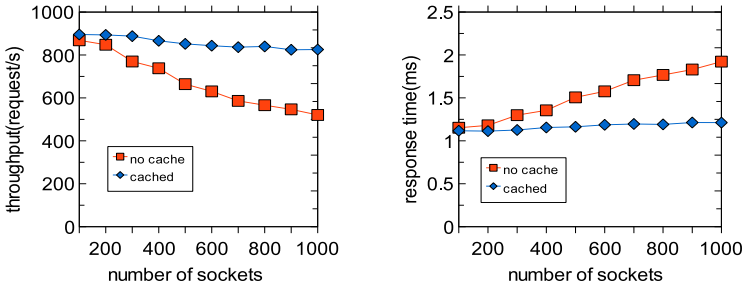
5.2 ウェブサーバの性能の低下

実際のアプリケーションの性能の低下を調べるため、Apache ウェブサーバのスループッ



(a) スループット　　(b) 応答時間

図 6　プロセス数を変更させた時のウェブサーバの性能



(a) スループット　　(b) 応答時間

図 7　ソケット数を変化させた時のウェブサーバの性能

トとレスポンス時間を測定した。ApacheBench ベンチマークツールをクライアントマシンで動作させた。ApacheBench はサーバマシンの VM 内で動作する Apache に HTTP リクエストを送る。リクエストする HTML ファイルのサイズは 50KB とした。上の実験と同様のフィルタリングルールで実験を行った。フィルタリング結果のキャッシュの効果を示すため、キャッシュを有効にした場合と無効にした場合について実験を行った。xFilter を用いなかった場合、スループットは毎秒 922 リクエスト、応答時間は 1.1ms であった。

まず、プロセス数を変えてウェブサーバの性能を測定した。図 6 はその結果を示しており、性能はプロセス数に比例して低下している。ただしキャッシュを有効にすると、5000 プロセスでもスループットの低下は 21%、レスポンスの低下は 33%であった。より現実的な 500 プロセスでは性能の低下は 4%であった。キャッシュを無効にすると、500 プロセスでさえスループットは 14%の低下、レスポンスは 16%の低下になった。これらの結果より、キャッシュはオーバーヘッドの削減に効果的であることが分かる。

次に、ソケット数を変えてウェブサーバの性能を測定した。スループットと応答時間は図 7 のようになり、性能の低下はソケット数に比例している。キャッシュを有効にすると、1000 ソケットでもスループットの低下は 10%、レスポンスの低下は 12%であった。100 ソケットでは、性能の低下は 3%であった。キャッシュを無効にすると、スループットの低下は 44%、レスポンスの低下は 77%になった。

6. 関 連 研 究

Amazon EC2 は security groups[1)] と呼ばれるファイアウォールを提供している。security groups は VMM 内で動作する外部ファイアウォールであり、ドメイン 0 に実装されていると思われる。このファイアウォールは対象 VM の外部に置かれているため、VM から攻撃されにくい。データセンタの出入り口に置かれているファイアウォールと違い、security groups は同一ホスト間のパケットフィルタも可能である。しかし、security groups は外部からの攻撃に対する受信パケット専用のファイアウォールなので、踏み台攻撃を防ぐことはできない。xFilter は内部からの踏み台攻撃に対する送信専用のパケットフィルタである。

フィルタリング結果のキャッシュは、ステートフル・パケット・インスペクション（SPI）に似ている。ファイアウォールの SPI は SYN フラグがセットされたパケットのみフィルタリングルールをチェックし、ステートテーブルに TCP コネクションの状態を保存する。コネクションが ESTABLISHED 状態になったら、コネクション内の全パケットを許可する。フィルタリング結果のキャッシュとの主な違いは、フィルタ結果のキャッシュはただの

キャッシュであるが、ステートテーブルはそうではないことである。フィルタリング結果のキャッシュが一杯になり使われているコネクションのエントリが削除されても、xFilter がパケットを許可するかどうか決めるために再び VM のメモリを解析することができる。一方、SYN flood 攻撃でステートテーブルがオーバーフローすると、使われているコネクションのパケットを間違って拒否してしまう可能性がある。

ident プロトコル[5] はパケットを誰が送信したかを尋ねるために使われる。あるホストが別のホスト内で動作する ident サーバに送信元と送信先のポート番号のペアを送ると、サーバはそのネットワークコネクションを利用しているプロセスの所有者を返す。踏み台攻撃が行われているとき、ident サーバは踏み台ホスト内で動作しているため攻撃を受ける可能性がある。さらに、このプロトコルはファイアウォールからではなくクライアントから利用することを想定している。xFilter は攻撃された VM 内のサーバプロセスに依存せずに、OS カーネルを直接調べる。

Chorus[11] や CAPERA[6] はカーネルモジュールの開発をサポートしている。これらの OS ではユーザプロセスとしてカーネルモジュールを実装し、修正なしにカーネルに埋め込むことができる。同様に、xFilter はヘルパー VM 内のユーザプロセスとしてモジュールを実装し、修正なしに VMM に埋め込むことができる。

7. ま と め

本稿では、VMM で動作するきめ細かいパケットフィルタ xFilter を提案した。xFilter を用いてパケットの送信元のプロセスやユーザを指定してパケットフィルタリングを行うことで、可能な限り踏み台攻撃の通信のみを制限することができる。ゲスト OS 内の情報は、VMのメモリを参照しカーネルの型情報を用いて解析することによって取得する。IDS も VMM 内で動作させることで、踏み台攻撃を検出してから送信元を特定するまでのタイムラグをできるだけ減らしている。さらに、検査結果をキャッシュしておくことで xFilter のオーバーヘッドを削減した。

参 考 文 献

1) Amazon, Inc. Amazon Web Services: Overview of Security Processes. http://aws.amazon.com/security/, 2009.
2) D.Donoho, A.Flesia, U.Shankar, V.Paxson, J.Coit, and S.Staniford. Multiscale Stepping-stone Detection: Detecting Pairs of Jittered Interactive Streams by Exploiting Maximum Tolerable Delay. In *Proc. Int. Symp. Recent Advances in Intrusion Detection*, pp. 17–35, 2002.
3) T.Garfinkel and M.Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *Proc. Network and Distributed Systems Security Symp.*, pp. 191–206, 2003.
4) T.He and L.Tong. Detecting Encrypted Stepping-stone Connections. *IEEE Trans. Signal Processing*, Vol.55, pp. 1612–1623, 2007.
5) M.Johns. Identification Protocol. RFC 1413, 1993.
6) Kenichi Kourai, Shigeru Chiba, and Takashi Masuda. Operating System Support for Easy Development of Distributed File Systems. In *Proc. Int. Conf. Parallel and Distributed Computing and Systems*, pp. 551–554, 1998.
7) Jr. N.Petroni and M.Hicks. Automated Detection of Persistent Kernel Control-flow Attacks. In *Proc. Conf. Computer and Communications Security*, 2007.
8) B. Payne, M. Carbone, and W. Lee. Secure and Flexible Monitoring of Virtual Machines. In *Proc. Annual Conf. Computer Security Applications*, pp. 385–397, 2007.
9) B.Payne, M.Carbone, M.Sharif, and W.Lee. Lares: An Architecture for Secure Active Monitoring Using Virtualization. In *Proc. Symp. Security and Privacy*, pp. 233–247, 2008.
10) N.Petroni, T.Fraser, J.Molina, and W.Arbaugh. Copilot – a Coprocessor-based Kernel Runtime Integrity Monitor. In *Proc. USENIX Security Symp.*, 2004.
11) M.Rozier, V.Abrossimov, F.Armand, IBoule, M.Gien, M.Guillemont, F.Herrman, C.Kaiser, S.Langlois, P.Léonard, and W.Neuhauser. Overview of the Chorus Distributed Operating System. In *Proc. Symp. Microkernels and Other Kernel Architectures*, pp. 39–69, 1992.
12) S. Staniford, J. Hoagland, and J. McAlerney. Practical Automated Detection of Stealthy Portscans. In *Proc. Conf. Computer and Communications Security*, 2000.
13) S.Staniford-Chen and L.Heberlein. Holding Intruders Accountable on the Internet. In *Proc. Symp. Security and Privacy*, pp. 39–49, 1995.
14) Trusted Computing Group. http://www.trustedcomputinggroup.org/.
15) Y.Zhang and V.Paxson. Detecting Stepping Stones. In *Proc. USENIX Security Symp.*, pp. 171–184, 2000.