日本ソフトウェア科学会第 26 回大会 (2009 年度) 講演論文集

Xen によるゲスト OS の解析に基づくパケットフィルタリング

安積 武志　光来 健一　千葉 滋

踏み台攻撃に対処するためにファイアウォールで通信を遮断する場合、ホストやポート単位での通信制御しか行うことができない。一方、踏み台攻撃が行われているホストで遮断すればきめ細かい制御ができるが、攻撃者によってセキュリティを無効化されてしまう危険性がある。そこで、サーバを仮想マシン (VM) を使って動かし、仮想マシンモニタ (VMM) できめ細かいパケットフィルタリングを行うシステム xFilter を提案する。VMM は VM から隔離されているため、xFilter は安全に動作することができる。VMM から VM 内の情報を取得するために、VM のメモリをマップしてゲスト OS のカーネルの型情報を用いて解析する。xFilter のオーバーヘッドを削減するために、送信するパケットをまとめて検査することでゲスト OS のメモリ解析の回数を減らす。また、TCP/IP 通信については検査結果をキャッシュすることで高速にフィルタリングを行う。

1　はじめに

サーバに脆弱性があると攻撃を受けて侵入され、他のホストへの踏み台攻撃に使われてしまう危険性がある。例えば、DDoS 攻撃を行うホストの 1 つとして使われたり、大量の SPAM メールを送るために使われたりする場合がある。踏み台にされたサーバの管理者は実際に攻撃を行ったわけではなくとも、結果的に攻撃行為の幇助を行っていたことによる責任を負うことになったり、信用を失うことになる。

攻撃者がサーバに侵入したことを検出することが望ましいが、それができなかったとしても、侵入されたサーバが他のホストへ攻撃を開始したらすぐに対処する必要がある。しかしサーバが行われているサイトのファイアウォール等で対処を行うと大雑把な通信制御しかできない。サーバ内の通信の詳細な情報を用いることができないため、ホスト単位やポート単位でしか通信の遮断ができないためである。一方、踏み台攻撃が置かれているサーバ内のファイアウォール等で対処を行えば、通信元のプロセスの情報などを用いてきめ細かい通信制御を行うことができる。しかし、攻撃者に管理者権限を奪われてしまうとファイアウォールのルールなどのセキュリティポリシを無効化されてしまう可能性があり、安全とはいえない。

我々はサーバを仮想マシン (VM) 内で動作させ、仮想マシンモニタ (VMM) できめ細かいパケットフィルタリングを行うシステム xFilter を提案する。VMM は VM から隔離されているため、攻撃者が VM 内に侵入したとしても xFilter を無効化することはできない。また、VM が送受信するパケットはすべて VMM を経由するため、VMM でフィルタリングすることができる。VMM から VM 内のゲスト OS の情報を使ってフィルタリングを行えるようにするために、xFilter はゲスト OS のメモリをマップして、カーネルの型情報を用いて解析する。これによって、通信元の特定のプロセスやユーザが行う通信に対してフィルタリングを行うことができる。

xFilter のオーバーヘッドを削減するために、ゲスト OS のメモリを解析する回数を減らすための工夫を

Packet filtering based on analyzing guest OSes by Xen.

Takeshi Azumi　Shigeru Chiba, 東京工業大学 数理・計算科学専攻, Dept. of Mathematical and Computing Science, Tokyo Institute of Technology,

Kenichi Kourai, 九州工業大学 情報創成工学研究系, Department of Department of Creative Informatics Kyushu Institute of Technology 科学技術振興機構, CREST, Japan Science and Technology Agency,

行った。パケットが送信された時に即座に検査するのではなく、キューにとっておいてある程度まとめて検査する。また、TCP コネクションを確立する時にパケットを検査した結果をキャッシュしておき、そのコネクションを流れるパケットはキャッシュを参照してフィルタリングを行う。xFilter の性能を測定する実験を行い、これらの工夫に一定の効果があることが分かった。

以下、2 章では踏み台攻撃に対する既存の対処法の問題点について述べる。3 章では VMM におけるパケットフィルタリングシステム xFilter を提案し、その実装について述べる。4 章では xFilter の性能を調べた実験の結果を示す。5 章で関連研究について触れ、6 章で本稿をまとめる。

2 踏み台攻撃に対する対処

踏み台攻撃を行う攻撃者は、特定のホストに対して大量のパケットを送ってサービス妨害攻撃を行ったり、更なる踏み台を求めて不特定多数のホストにポートスキャンを行うなどの通常では行われない行為を行うことが多い。このような異常なネットワークアクセスはファイアウォール等で検出可能であり、踏み台攻撃自体の検出は比較的容易である。

一方、踏み台攻撃が検出された後の一つの対処法は、ファイアウォール等で対象ホストからの通信を制限することである。例えば、メールサーバに侵入されて、攻撃者が他のホストの 25 番ポートに対してスキャンを行っているという状況を考える。ファイアウォールで行える最も簡単な対処としては、踏み台攻撃を行っているホストからの通信を全て遮断してしまうことである。また、このように特定ポートだけを利用した踏み台攻撃が行われている場合には、25 番ポートへの通信だけを遮断するという方法も考えられる。

この対処法は攻撃を受けたホストの外側で行うため、攻撃者が回避するのは難しく、安全性は高い。しかし、通信制限に利用できる情報はパケットに含まれる情報だけであり、IP アドレス単位やポート単位といった大雑把な制限しかできず、問題のない通信までも制限されてしまう。上の例では、IP アドレス単位で通信を制限するとメールサーバ全体がサービスを提供できなくなり、ポート単位で制限してもメールを外部に送ることができなくなってしまう。

なるべく踏み台攻撃で行われている通信だけを制限する対処法としては、踏み台攻撃を行っているホスト内部で通信を制限する方法が考えられる。ホスト内部の OS の情報を用いることで、プロセスやユーザを意識してきめ細かい通信制限を行うことが可能になる。例えば、メールサーバから上記のような踏み台攻撃が検出された時、その通信を行ったプロセスやユーザを調べることは容易である。ユーザ A の所有するプロセスから踏み台攻撃が行われていれば、そのユーザアカウントから侵入されたと判断し、ユーザ A もしくはユーザ A の特定のプロセスからの通信だけを制限すればよい。このような対処を行えば、メール配送サービスを停止させることなく、25 番ポートへのスキャンだけを制限することができる。

しかし、すでに攻撃を受けているホストでこのような対処を行っても、通信制御を無効化されてしまう可能性があり、安全とはいえない。一般にセキュリティポリシの設定には管理者権限が必要となるが、攻撃者に管理者権限を奪われてしまうと、セキュリティポリシを書き換えられてしまう。例えば、特定のプロセスからの通信を制限するようにファイアウォールのルールを追加したとしても、攻撃者はそのルールを削除して踏み台攻撃を続けることができる。

3 xFilter

3.1 概要

安全かつ、きめ細かい通信制限を可能にするために、我々は仮想マシン（VM）を使ってサーバを動作させ、仮想マシンモニタ（VMM）でフィルタリングを行う xFilter を提案する。VMM は VM を動作させるための基盤となるソフトウェアであり、VM からの通信は VMM を介して行われる。VMM は VM から隔離されているため、VM に侵入した攻撃者が VMM で行うパケットフィルタリングを無効化することは難しい。一方、VMM からは VM のメモリ等の情報が低レベルではあるが参照できるため、VM 内で動作する OS（ゲスト OS）内部の情報を利用する

ことができる。

xFilter はゲスト OS 内部の通信の情報を VMM 経由で取得し、プロセスやユーザの情報を用いて VMM 内できめ細かいパケットフィルタリングを行う。VM のメモリのバイト列とゲスト OS 内部のデータ構造の間のセマンティックギャップを埋めるために、xFilter はゲスト OS のデータ構造の型情報を用いて VM のメモリを解析する。これにより、従来では利用することができなかったゲスト OS 内部の通信の情報を VMM から取得し、パケットフィルタリングに用いることが可能になる。

xFilter によるパケットフィルタリングは以下のように行われる。踏み台攻撃により行われている通信が検出されたらまず、VMM 経由でゲスト OS の情報を参照して検出された通信を行っているプロセスやユーザを特定する。次にそのプロセスまたはユーザの行う通信を禁止するフィルタリングルールを xFilter に追加する。2 章のメールサーバの例では、ユーザ A が踏み台攻撃を行っていることがわかれば、ユーザ A から 25 番ポートへの通信を禁止するルールを追加する。xFilter がパケットを検査する際には、ゲスト OS のどのユーザが送信したパケットか調べ、例えばユーザ A が送信したパケットであれば送信を拒否する。

3.2 システム構成

VMM として Xen [1]、ゲスト OS として Linux を使用して xFilter を実装した。Xen において VM はドメインと呼ばれ、管理用の特権 VM であるドメイン 0 と一般の VM であるドメイン U が存在する。ドメイン 0 からはドメイン U のメモリを参照することが可能であり、物理 NIC に対するネットワーク処理も行うことから、xFilter はドメイン 0 上で実装した。

xFilter のシステム構成は図 1 のようになっている。xFilter はドメイン 0 上のプロセスとして動作するメモリ解析部と、ドメイン 0 のカーネル内部で動作するフィルタリング部からなる。Xen において、ドメイン U は netfront と呼ばれるネットワークデバイスドライバを使ってパケットを送信する。netfront は Xen が提供する機構を使ってドメイン 0 のカーネル内のデバイスドライバである netback と通信する。netback は物理 NIC にアクセスするための実デバイスドライバにパケットを渡すことでネットワーク送信を行う。

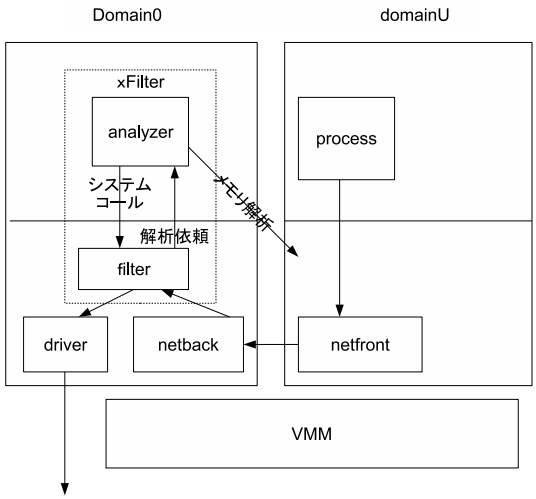

図 1 xFilter のシステム構成

xFilter では netback と実デバイスの間でドメイン U からのパケットをフィルタリングする。netback がパケットを受け取るとフィルタリング部を呼び出し、フィルタリング部はメモリ解析部に解析を依頼する。メモリ解析部はゲスト OS 内のメモリを解析して、受け取ったパケットを送信したプロセスやユーザの情報を取得し、新たに追加したシステムコールを用いてその情報をフィルタリング部に通知する。フィルタリング部では、その情報と受け取ったパケットのヘッダ情報の組をフィルタリングルールと比較し、送信が禁止されていればパケットを破棄する。

3.3 ゲスト OS のメモリ解析

ゲスト OS のメモリを解析するために、ドメイン 0 からドメイン U のメモリを参照して、ゲスト OS のデータ構造を操作する機構 [4] を利用した。この機構を用いると、ドメイン U の仮想アドレスから VMM が管理するメモリフレーム番号を取得することができる。内部的には、ドメイン U のページテーブルを参照して仮想アドレスからメモリフレームを取得している。取得したメモリフレームをドメイン 0 のプ

ロセスのアドレス空間にマップすることで、ドメイン 0 のプロセスからドメイン U のメモリにアクセスすることができる。そして、ゲスト OS のカーネルのデバッグ情報から得た型情報を利用することで、ドメイン U のメモリ内のデータ構造を解析し、そのポインタをたどっていくことで必要な情報を取得することができる。

この機構を用いて全てのプロセスの通信情報を取得する手順を説明した図が図 2 である。まず Xen の機能を使ってドメイン U の仮想 CPU の GS レジスタを取得し、カレントプロセスのプロセス情報を管理している `task_struct` 構造体のアドレスを取得する。`task_struct` 構造体はプロセス ID 順にリング状につながっているため、これを順番に見ていけば全てのプロセスを調べることができる。この構造体からプロセス ID やユーザ ID の情報を取得することができる。また、この構造体はオープンしているファイルを管理している `file` 構造体の配列へのポインタを持っている。`file` 構造体にはファイルとソケットの両方が格納されうるが、メンバ f_op がグローバル変数 socket_file_ops のアドレスと一致すればソケットと判断できる。このグローバル変数のアドレスはカーネルをコンパイルするときに作られる System.map から取得した。`file` 構造体からソケットを管理している `sock` 構造体までポインタをたどることができ、この構造体に通信のポート番号や IP アドレスなどの情報が格納されている。

3.3.1 解析範囲の絞込み

ドメイン U のメモリにアクセスするためには、アクセスしたいデータがあるメモリページをマップしてアクセスするという作業が必要となるため、すべての通信を調べあげるとページをマップするオーバーヘッドが大きくなる。各プロセスが使っているソケットの通信の情報を取得するには、7 ページ以上マップしなければならず、これをすべてのプロセスについて行わなければならない。

ドメイン U のメモリへのアクセス回数を減らすために、すべての通信を調べるのではなく、まず、`task_struct` 構造体を参照してプロセス ID やユーザ ID がフィルタリングルールとマッチするかどうか調べる。マッチしなければそのプロセスの通信については調べない。この絞込みにより、ルールと一致しなければそのプロセスについては 1 回のメモリアクセスですむ。

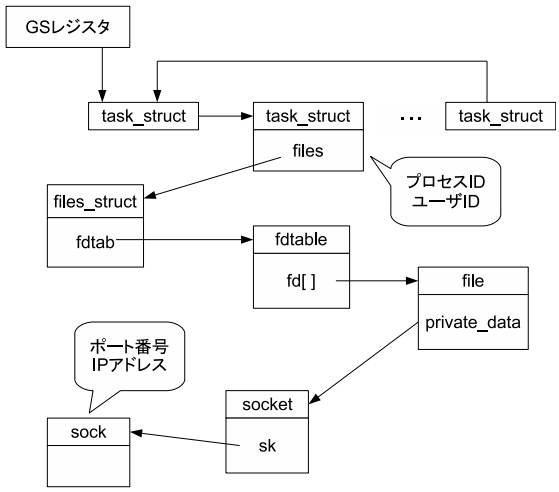

図 2 構造体の追跡

3.3.2 情報の一覧表示機能

xFilter は踏み台攻撃として検出されたパケットの送信元のプロセスやユーザの特定を支援するために、ゲスト OS の全プロセスのプロセス ID、ユーザ ID、通信情報の一覧を表示する機能を提供している。管理者はこの機能を使って攻撃元のプロセスを調べ、同一のプロセスからの攻撃であればそのプロセスからの通信を禁止するフィルタリングルールを追加すればよい。一方、同一のプロセスからの攻撃でなくても同一のユーザからの攻撃だった場合は、そのユーザからの通信を禁止するルールを追加すればよい。

3.4 パケットフィルタリング

ドメイン 0 のカーネル内のフィルタリング部はドメイン U の netfront ドライバからドメイン 0 の netback ドライバにパケットが到着したらそのパケットのヘッダを解析する。ヘッダに含まれている送信元と送信先の IP アドレスとポート番号の情報から、ゲスト OS のメモリ解析によってどのプロセスまたはユー

ザが送信したパケットかを特定し、これらの情報を使ってパケットの検査を行う。しかし、この検査をパケット毎に行うと大幅な性能低下が避けられない。検査のオーバーヘッドを削減するために、二つの手法を実装した。

3.4.1 一括検査

ゲスト OS のメモリ解析の回数を減らすためにパケットが到着してもすぐに検査せず、ある程度キューにためておいてまとめて検査する。netback ドライバはパケットを受け取ったときすぐに検査を行う代わりに、フィルタリング部のキューにパケットを入れる。そしてフィルタリング部は一定時間毎にこのキューにたまったパケットをまとめて検査する。これにより、パケット到着毎に行っていたゲスト OS のメモリ解析を、このキューにとっておいたパケット全てに対して行えるようになる。検査の間隔は xFilter の解析部が管理しており、指定した時間毎にゲスト OS のメモリを解析し、取得した情報をフィルタリング部に通知するためのシステムコールを呼び出す。

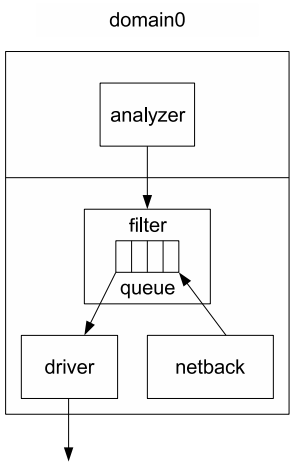

図 3 一括検査のしくみ

3.4.2 検査結果のキャッシュ

同一プロセスによる同一の IP アドレスとポート番号への通信を何回も検査せずに済ませるために、フィルタリング部ではパケットの検査結果をキャッシュする。フィルタリング部はパケットを受け取った時にまずキャッシュをチェックし、検査結果がキャッシュにあって送信が許可されていればキューに入ることなく即座に送信処理を行う。特に TCP 通信については、同一コネクションのパケット群には同じ検査結果を適応することができる。そこで、TCP に対してフィルタリングを行う時は TCP ヘッダを解析してパケットのフラグを見る。SYN フラグがたっていればコネクションを確立しようとしていると考えられるため、ゲスト OS のメモリを解析して通信を許可するかどうかを判断する。そして、パケットの IP アドレスとポート番号をその検査結果とともにキャッシュする。FIN フラグがたっていればコネクションの終了を意味するので、キャッシュからエントリを削除する。キャッシュ機能により、一度確立した TCP コネクションを使って送信処理を行うときにゲスト OS のメモリ解析を行う必要がなくなる。キャッシュをチェックすること以外は通常の送信処理と同じ処理を行うため、キャッシュをチェックするオーバーヘッドしか発生せず、これはゲスト OS のメモリ解析に比べてかなり小さい。

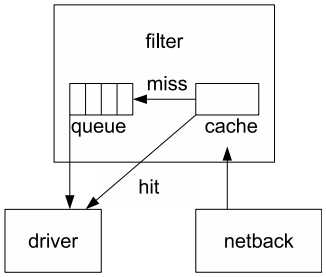

図 4 キャッシュ機能

4 実験

xFilter のオーバーヘッドを調べるために、 httperf ベンチマーク [3] を用いて通信の性能を測定した。xFilter を動作させるサーバは、Athlon 64 Processor 3500+ の CPU を一基、メモリ 1GB、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 3.1.0 、VMM 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

512MB、ドメイン U にはメモリを 256MB 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 を一基、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。このマシンでは httperf 0.9.0 を動かし、ウェブサーバ上にあるサイズ 3918 バイトの HTML ファイルにリクエストを送らせた。これらの計算機は、ギガビットスイッチで接続した。

4.1 xFilter の性能

パケットの送信が許可されている状態での性能を調べるために、フィルタリングにマッチしないルールを 1 つだけ設定して、測定を行った。xFilter を使わない場合、フィルタリング間隔を 50ms、100ms、200ms、300ms にした場合の 5 通りについて、リクエストを送るために TCP コネクションの確立を開始してからレスポンスを受け取って TCP コネクションを切断するまでの時間を計測した。実験結果は表 1〜3 のとおりである。

表 1 毎秒 100 リクエスト時のリクエスト処理時間 (ms)

検査間隔	最小	平均	最大	中央値
xFilter なし	0.3	0.4	10.6	0.5
50ms	70.4	110.9	159.2	110.5
100ms	120.6	187.0	3210.5	180.5
200ms	231.4	510.6	2011.9	471.5
300ms	330.7	1241.5	4638.5	1199.5

表 2 毎秒 150 リクエスト時のリクエスト処理時間 (ms)

検査間隔	最小	平均	最大	中央値
xFilter なし	0.3	0.4	7.6	0.5
50ms	66.9	106.3	147.7	106.5
100ms	120.5	188.6	721.3	183.5
200ms	232.6	1160.5	5482.8	868.5
300ms	333.3	1569.8	4972.2	1475.5

表 3 毎秒 200 リクエスト時のリクエスト処理時間 (ms)

検査間隔	最小	平均	最大	中央値
xFilter なし	0.3	0.4	5.5	0.5
50ms	65.8	137.9	3136.3	108.5
100ms	126.1	328.4	934.0	316.5
200ms	221.4	2557.3	9553.7	1981.5
300ms	333.0	2631.6	6249.9	2711.5

xFilter を用いない場合と比べると、xFilter を用いることによってリクエストの処理時間が非常に長くなってしまっていることが分かる。しかし検査間隔を短くするほど、送信せずにキューにとっておく時間の平均が短くなるため、一つのリクエストを処理するのにかかる時間は短くなった。検査間隔を長くすると、タイムアウトするパケットがでてきてパケットの再送が行われるためリクエストの処理非常に長い時間がかかる場合があった。レートをあげた場合も同様で、レートが高くなるほどタイムアウトするパケットが多くなった。しかし、検査間隔が 50ms ならばレートが 200 でもごく稀にタイムアウトする程度で、スループットには影響がなかった。

4.2 オーバーヘッドの内訳

xFilter のオーバーヘッドの内訳を調べるために毎秒 100 リクエストの場合についてさらに実験を行った。ゲスト OS のメモリ解析にかかる時間を計測するために、検査間隔を 50ms にしたときにメモリ解析を行わない場合の性能を測定した。また、パケットをキューにためることによるオーバーヘッドを調べるために、検査間隔を 0ms にして連続してフィルタリング処理を行った場合の性能も測定した。この場合、キューにたまっているパケット群の処理が終わると、即座に、その処理中にキューにたまったパケット群の処理を開始することになる。この実験でもゲスト OS のメモリ解析は行わないようにした。実験結果は表 4 の通りである。

ゲスト OS のメモリ解析によるオーバーヘッドは平均 15.4ms であると考えられる。このオーバーヘッド

表 4　様々な条件でのリクエスト処理時間 (ms)

解析	間隔	最小	平均	最大	中央値
xFilter なし		0.3	0.4	10.6	0.5
なし	0ms	0.7	13.9	70.3	10.5
なし	50ms	60.2	95.5	178.9	94.5
あり	50ms	70.4	110.9	159.2	110.5

が全体に占める割合はそれほど大きくないため、パケットが到着するたびにフィルタリング処理を行ったほうが性能がよくなるのではないかと思われるかもしれない。しかし、毎秒 100 パケットであってもメモリ解析にかかる時間は平均 1.54 秒かかるため、パケットを処理しきれなくなる。

パケットを検査する間隔を極限まで短くしたときのオーバーヘッドは 13.5ms であることが分かった。このオーバーヘッドにはフィルタリング処理中に到着したパケットがキューで待たされる時間が含まれている。

4.3　キャッシュの利用による性能改善

検査結果をキャッシュするようにした場合の性能を測定した。キャッシュはゲスト OS のメモリを解析するオーバーヘッドと、パケットをキューにためることによるオーバーヘッドを削減する。メモリ解析によるオーバーヘッドは 4.2 節ですでに調べたので、キューにためるオーバーヘッドがどのくらい削減できるかを調べるためにメモリ解析は行わないようにした。実験結果は表 5 の通りである。

表 5　キャッシュを使った場合のリクエスト処理時間 (ms)

	最小	平均	最大	中央値
xFilter なし	0.2	0.4	31.1	0.5
キャッシュあり	4.3	38.3	90.3	37.5
キャッシュなし	60.2	95.5	178.9	94.5

キャッシュを用いることで 1 リクエストを処理する時間を平均 57.2ms 削減することができた。この実験では 1 リクエストごとに 1 コネクションを確立しているため、TCP コネクションを確立する最初のパケットについてはキューにためて処理することになる。コネクションを確立するのにかかる時間とリクエストを送ってからレスポンスが帰ってくるまでの時間の内訳は表 6 のようになった。

表 6　リクエスト処理時間の内訳 (ms)

	コネクション	レスポンス
xFilter なし	0.2	0.3
キャッシュあり	35.7	2.6
キャッシュなし	35.8	59.7

コネクションの確立にかかる時間が変わらないのは、最初の SYN + ACK パケットはキャッシュにヒットせず、キューで待たされるためである。一方、コネクションが確立した後にリクエストを送るパケットはキャッシュにヒットして即座に送信処理が行われるためにレスポンスは大幅に向上した。1 コネクションで多くのパケットをやり取りする場合には、キャッシュを使うことにより xFilter のオーバーヘッドがかなり削減されると思われる。xFilter を用いない場合との 2.3ms の差は、キャッシュがあるか調べるためにかかる時間であると考えられる。

5　関連研究

Livewire [2] は、VMM から VM 内のゲスト OS を監視することで OS に依存せずに侵入を検知するシステムである。侵入検知システム (IDS) を VM の外で動作させることで、VM 内に侵入した攻撃者から IDS 自体を守ることができる。VM に割り当てたメモリの内容を解析することで、ゲスト OS のプロセスの状態を取得するところは本研究と同じである。Livewire は侵入を検知するシステムであるが、xFilter は侵入された後に行われる踏み台攻撃による通信の制御を行うシステムである。

FreeBSD の ipfw では、IP アドレスやポート番号などに加えて、パケットを送受信するユーザやグループの情報を使ってフィルタリングできる。一方、Linux の iptables では、パケットの送信元のユーザ、グループ、プロセス、セッションの情報を使ってフィルタリングを行うことができるが、受信先については指定す

ることができない。xFilter ではこれらと同様のことをゲスト OS の外側の VMM から行うことを可能にしている。

xFilter で行っている検査結果のキャッシュはファイアウォールのステートフルインスペクションに似ている。ステートフルインスペクションでは TCP の SYN パケットがファイアウォールに到着したときはルールベースでチェックを行い、許可されたパケットに関する情報はステートテーブルに追加する。同一セッションのそれ以降のパケットについてはルールベースのチェックではなくステートテーブルを使ってチェックを行う。ステートフルインスペクションはセッションの概念がない UDP 等に対しても、パケットの状態から通信状態を追跡する仮想的なセッションをステートテーブル内に生成している。xFilter でも同様の技術を用いることで UDP の場合にも検査結果をキャッシュできると考えられる。

6 まとめと今後の課題

VMM でゲスト OS と同様のきめ細かいパケットフィルタリングを可能にするシステム xFilter を提案した。xFilter を用いてパケット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、踏み台攻撃を行われた場合でも通信を制限する範囲を最小限に抑えることができる。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メモリを解析することによって取得する。さらに、ある程度のパケットをまとめて検査したり、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。実験の結果から、これらのオーバーヘッド削減の効果はあったものの、xFilter のオーバーヘッドはまだ大きいことが分かった。

今後の課題は、xFilter のオーバーヘッドをさらに削減することである。現在はメモリ解析部をユーザランドプロセスとして実装しているため、メモリをマップするオーバーヘッドが大きい。この部分を VMM 内に実装することで VM のメモリを直接参照することができるようになり、このオーバーヘッドを削減できると考えている。また、メモリ解析部とフィルタリング部のやりとりにシステムコールを使っていることもオーバーヘッドの一因だと考えられる。

謝辞 研究に関して適切な助言をしていただいた千葉研究室の方々に心より感謝いたします。

参 考 文 献

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 164–177, New York, NY, USA, 2003. ACM Press.

[2] Tal Garfinkel and Mendel Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proc. Network and Distributed Systems Security Symposium*, February 2003.

[3] David Mosberger and Tai Jin. httperf-a tool for measuring web server performance. *SIGMETRICS Perform. Eval. Rev.*, Vol. 26, No. 3, pp. 31–37, 1998.

[4] 田所秀和, 光来健一, 千葉滋. 仮想マシン間にまたがるプロセススケジューリング. 情報処理学会論文誌:コンピューティングシステム. ACS 23.