Vol.2010-ARC-189 No.12
Vol.2010-OS-114 No.12
2010/4/22

仮想マシンモニタによるきめ細かいパケットフィルタリング

安積武志[†1] 光来健一[†2,†3] 千葉滋[†1]

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。本稿では、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）でゲスト OS の情報を利用して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 からゲスト OS 内の情報を取得するために、xFilter はゲスト OS のカーネルの型情報を用いて VM のメモリを解析する。また、フィルタモジュールの開発を用意にするために、xFilter はフィルタモジュールを専用の VM 内で動かすことも可能にしている。フィルタモジュールを VMM 内で動かした場合、オーバーヘッドが非常に小さいことを確認した。

Fine-grained Packet Filtering in a Virtual Machine Monitor

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

When intruders attack other hosts from a victim host, an external firewall only allows coarse-grained packet filtering by IP addresses or ports. In contrast, an internal firewall at a victim host allows fine-grained packet filtering but may be invalidated by intruders. This paper proposes xFilter, which is a secure and fine-grained packet filter in a virtual machine monitor (VMM). To use the information on guest operating systems (OSes), xFilter analyses the memory of guest OSes in virtual machines (VMs) using their type information. In addition, xFilter enables running filter modules at a dedicated VM to make the development easier. We confirmed that the overhead of xFilter was negligible when a filter module runs in a VMM.

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、他のホストへの攻撃の踏み台に使われてしまう可能性がある。例えば、 DDoS 攻撃を行うホストの 1 つとして使われたり、大量のスパム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ても、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、サーバが動作しているサイトのファイアウォールなど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、踏み台攻撃が行われているサーバ内のファイアウォールなどで対処を行えば、通信元のプロセス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 で確実にフィルタリングを行うことができる。xFilter は VMM から VM 内のゲスト OS のメモリを参照して、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。複雑なゲスト OS のメモリ解析を行うフィルタモジュールの開発を用意にするために、xFilter はフィルタモジュールを専用 VM 内で動かすこともできる。

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

†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

Vol.2010-ARC-189 No.12
Vol.2010-OS-114 No.12
2010/4/22

におけるパケットフィルタリングシステム xFilter を提案し、4 章でその実装について述べる。5 章では xFilter の性能を調べた実験の結果を示す。6 章で関連研究について触れ、7 章で本稿をまとめる。

2. 踏み台攻撃の脅威

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信してサービスの妨害を行ったり、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容易である。

一方、踏み台攻撃が検出された後の 1 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う最も簡単な対処法としては、踏み台攻撃を行われているホストからの通信をすべ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の場合には、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おくれなくなってしまう。

なるべく踏み台攻撃を行っ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とき、その通信を行ったプロセスやユーザを特定することは容易である。ユーザ A の所有する特定の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、メールの配送サービスを停止することなく、25 番ポートへのスキャンだけを制限することが出来る。

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。

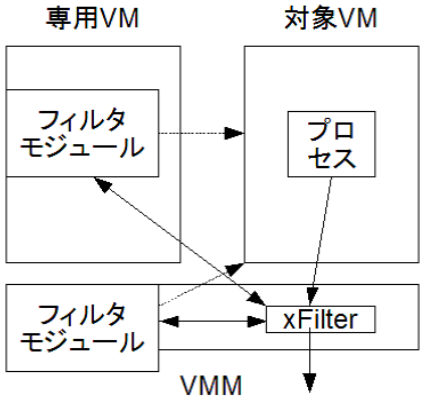

図 1 xFilter

3. xFilter

安全かつきめ細かい通信制御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、図 1 のように仮想マシンモニタ (VMM) でパケット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 を介して行われるため、確実にフィルタリングを行うことができる。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 の機能を用いて取得し、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 VMM からは利用することのできなかったゲスト OS 内部の通信の情報を、パケットフィルタリングに用いることが可能になる。

xFilter が行うパケットフィルタリングにはゲスト OS の様々な情報を用いることが可能であるため、フィルタリングアルゴリズムをフィルタモジュールとしてモジュール化している。フィルタモジュールは VMM に到着したパケットの情報を受け取り、フィルタリングルールに基づいてそのパケットを通す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。

通常、フィルタモジュールは VMM 内で動作させるが、開発時などは専用 VM で動作させることも可能である。VMM 内で動作させるとゲスト OS のメモリを直接参照できるためフィルタリング性能はよいが、フィルタモジュールにバグがあった場合、システム全体がクラッシュする危険性がある。フィルタモジュールは複雑なゲスト OS のメモリ解析を行う必要があるため、最初からバグのないフィルタモジュールを作成するのは難しい。一方、専用 VM でプロセスとして動作させると、フィルタモジュールがクラッシュしても立ち上げ直すだけでよい。その代わり、ゲスト OS のメモリを参照するために VMM の機能を利用してメモリマップを行ったり、VM の切り替えを伴ったりするため、フィルタリング性能は悪くなる。xFilter では性能と開発のしやすさのトレードオフをとれるようにしており、フィルタモジュールを専用 VM で開発して、完成したらそのまま VMM に組み込むことができる。

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、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。

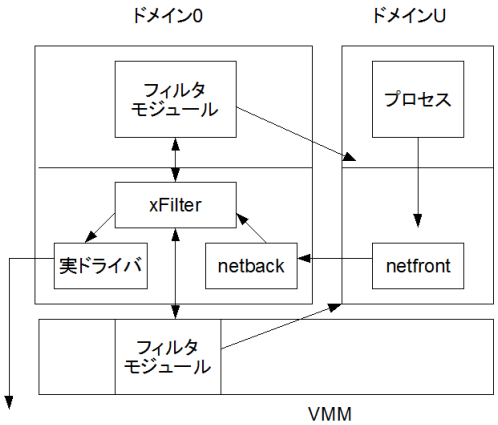

図 2 xFilter のシステム構成

4. 実　　装

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、管理者の特権 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。

4.1 システム構成

xFilter のシステム構成は図 2 のようになっている。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渡すことでネットワーク送信を行う。xFilter は netback と実デバイスドライバの間でドメイン U からのパケットをフィルタリングする。

xFilter が netback からパケットを受け取ると VMM またはドメイン 0 のユーザランドのフィルタモジュールを呼び出す。VMM 内のフィルタモジュールを呼び出す際にはハイ

Vol.2010-ARC-189 No.12
Vol.2010-OS-114 No.12
2010/4/22

パーコールが使われる。ユーザランドのフィルタモジュールは定期的にシステムコールを呼び出しており、xFilter にパケットが到着しているかどうかを返り値として返す。フィルタモジュールは、ゲスト OS 内のメモリを解析し、パケットを送信したプロセスやユーザの情報を取得する。次に、取得した情報を基にポリシに従って送信を許可するか否かを判定する。その結果を xFilter に返し、xFilter は受け取った結果に従いパケットを送信または破棄する。

4.2 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 U の仮想アドレスに対応するマシンアドレスを見つける。まず、ドメイン U のページテーブルを参照して仮想アドレスを擬似物理アドレスに変換し、擬似物理アドレスからマシンメモリフレーム番号に変換する。VMM 内では得られたマシンメモリフレーム番号から計算できるマシンメモリアドレスを用いることで、ドメイン U のメモリに直接アクセスできる。ドメイン 0 からは VMM の機能を用いてマシンメモリフレームをプロセスのアドレス空間にマップすることで、ドメイン U のメモリに直接アクセスできるようになる。次に、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。

このようにしてすべてのプロセスの通信情報を取得する手順が図 3 に示されている。プロセスの情報は task_struct 構造体が管理しており、プロセス ID 順で次になるプロセスの task_struct 構造体へのポインタを持っている。これらは環状にリンクしているため、1 つの task_struct 構造体からすべてのプロセスの情報を参照することができる。この構造体からプロセス ID やユーザ ID などの情報を取得することができる。xFilter ではまず、 init_task の task_struct 構造体のアドレスを参照する。init_task はプロセス ID が 0 のプロセスで、このプロセスを管理する task_struct 構造体のアドレスは静的に決定される。こ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。

また task_struct 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も持っている。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であると判断できる。socket_file_ops のアドレスも System.map から取得できる。file 構造体に格納されているデータがソケットだった場合、さらに 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る。この構造体に通信の IP アドレスやポート番号などの情報が格納されている。

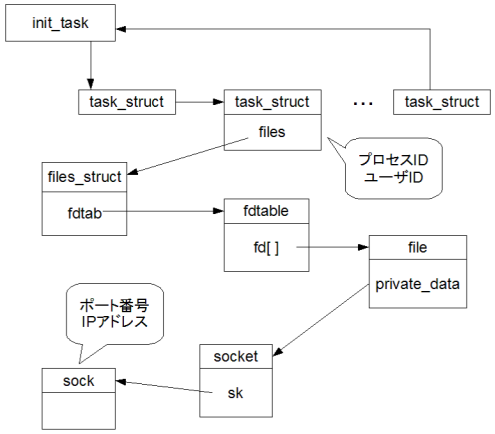

図 3 構造体の追跡

VMM からゲスト OS の task_struct のリストを操作する場合、ゲスト OS のカーネルが task_struct のリストを操作していないか確認する必要がある。xFilter は task_struct のリストのロックをチェックすることで一貫性を保ってゲスト OS の task_struct 構造体を操作する。task_struct 構造体はスピンロックを使って排他制御を行っており、そのアドレスは System.map から取得できる。ゲスト OS の task_struct 構造体にアクセスするときに、ゲスト OS のカーネルが task_struct のリストを操作していないことを保証するために、ロックが取られていないか確認する。ロックが取られている場合はゲスト OS カーネルが操作していると判断し、何もせずに終了する。

4.3 キャッシュ機能

フィルタリングのオーバーヘッドを削減するため、キャッシュ機能を実装した。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為に、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とき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ればメモリ解析を行わずに即座に処理を行う。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用することができる。そこで、TCP に対してフィルタリングを行う

Vol.2010-ARC-189 No.12
Vol.2010-OS-114 No.12
2010/4/22

ときは TCP ヘッダを解析してパケットのフラグを調べる。SYN フラグが立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 フラグが立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 のメモリ解析のオーバーヘッドに比べて小さい。

4.4 一括検査

ドメイン 0 でフィルタリングモジュールを動作させる場合、呼び出しのオーバーヘッドおよびゲスト OS のメモリ解析のオーバーヘッドが非常に大きいため、パケットが到着してもすぐに検査せず、ある程度キューにとっ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にすぐに検査を行う代わりに、キューにパケットを入れる。そして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、このキューに取っておいたパケットすべてに対して行えるようになる。

5. 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、Intel Core i7 860 の CPU を一基、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。

5.1 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、毎秒 100、200、300、400、500 のそれぞれのリクエストレートについて、100 秒間リクエストを送信した場合の性能を測定した。xFilter を使わない場合と xFilter を使う状態とで、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。

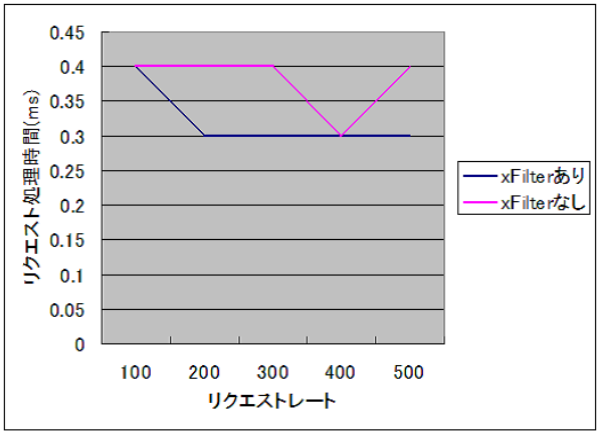

図 4 リクエスト処理時間

実験結果は図 4 のようになった。xFilter を毎秒 500 リクエストまでは xFilter を使うことによってリクエストの処理時間に影響はなかった。これ以上レートを上げて実験を行うとすぐにファイルディスクリプタがオーバーフローしてしまうため、ごく短時間しか実験できない。

リクエストレートが毎秒 100 の時に、フィルタモジュールの実行にかかる時間を調べた。この実験はでは VMM 内のフィルタモジュールを呼び出すハイパーコールの実行にかかる時間を測定した。その結果、フィルタモジュールの実行時間は平均で 19 μ s であり、最大でも 178 μ s であった。そのため、0.1ms 単位でしか測定できない httperf では xFilter のオーバーヘッドはほとんど影響しない。

5.2 ソケット構造体

フィルタリングルールにマッチするプロセスが存在しているが、フィルタリングされるパケットは送信していない状態で xFilter の性能を測定した。ルールにマッチしなければプロセス構造体のページのみを解析するが、マッチした場合はポインタをたどることでプロセスのすべてのソケット構造たのページも解析する必要がある。解析する必要があるソケットの

Vol.2010-ARC-189 No.12
Vol.2010-OS-114 No.12
2010/4/22

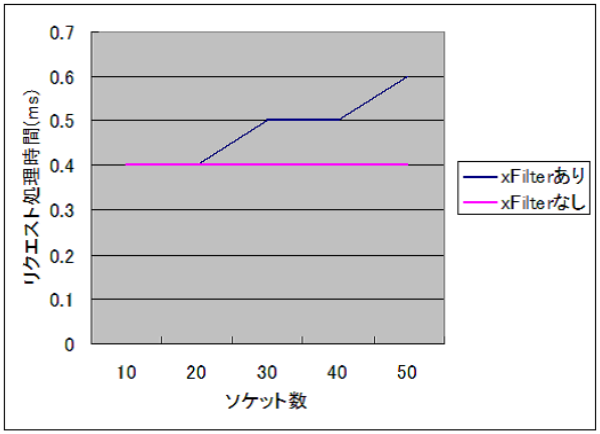

図 5 リクエスト処理時間

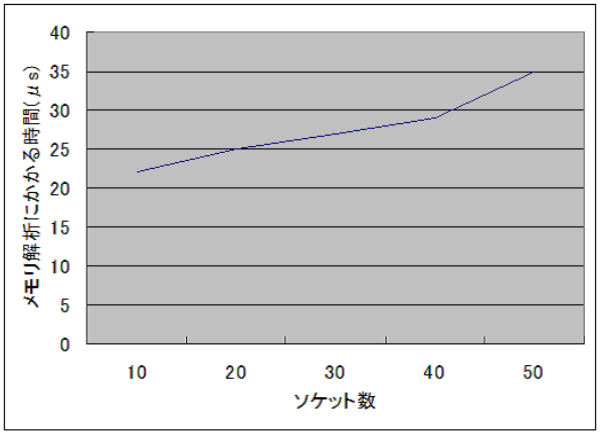

図 6 メモリ解析にかかる時間

表 1 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
xFilter なし	0.2	0.4	2.0
キャッシュあり	0.2	0.4	0.8
キャッシュなし	0.3	0.6	10.1

数を 10〜50 まで変えて、ウェブサーバのリクエスト処理時間を測定した。リクエストレートは毎秒 100 とした。

結果は図 5 のようになった。解析すべきソケットの数が増えるにつれて、ウェブサーバのリクエスト処理時間も若干増えている。一方、表 6 はメモリ解析にかかる時間を示している。ソケット数が増加するにつれて解析時間も長くなっているが、高々数十μs である。

5.3 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を削減する。リクエストレート 100、ルールにマッチするソケット数を 50 として、キャッシュを使う場合と使わない場合の性能を測定した。実験結果は表 1 のようになった。キャッシュを使わない場合は xFilter を使わない場合と繰らば手 0.2ms のオーバーヘッドがあるが、キャッシュがあればオーバーヘッドはない。

5.4 ドメイン 0 でのフィルタリングモジュール実行時の性能

ドメイン 0 でフィルタリングモジュールを動作させた場合、開発のしやすさを重視しているため性能が犠牲になる。どの程度の性能が得られるか調べるために、5.1 と同様の実験を行った。結果は表 7 のようになり、VMM 内でフィルタリングモジュールを動作させる場合と比べると尾 p-バーヘッドが大きいことが分かる。しかし、開発時であれば問題ない性能低下だと考えられる。

6. 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やグループ、プロセス、セッションの情報を使ってフィルタリング

Vol.2010-ARC-189 No.12
Vol.2010-OS-114 No.12
2010/4/22

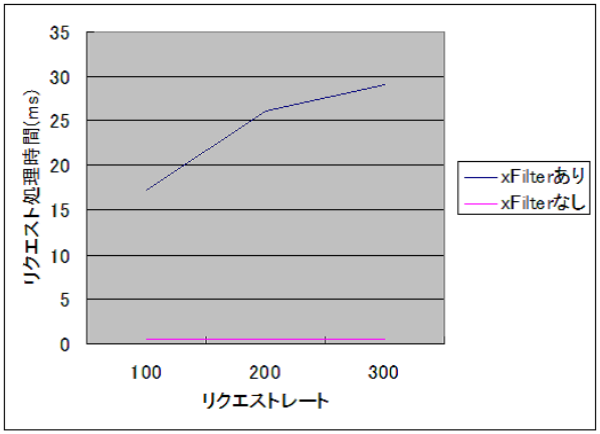

図 7 リクエスト処理時間

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 などに対しても、パケットの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。

7. ま と め

VMM でゲスト OS と同様に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、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解析することによって取得する。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。また、xFilter を安全に動作させるためにユーザランドプロセスとして実装を行った。

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.