仮想的な分散監視環境による安全な侵入検知アーキテクチャ

光来 健一[†] 千葉 滋[†]

本稿では分散システムにおける安全な侵入検知を実現する仮想的な分散監視環境 *HyperSpector* を提案する．分散化された侵入検知システム（IDS）は分散システムを守ることができる一方，それ自身の脆弱性が分散システム全体の危険性を増大させる可能性がある．HyperSpector は仮想化技術を用いて分散 IDS を監視対象のサーバから分離することでこの問題を解決する．IDS およびサーバはそれぞれ *IDS VM*，**サーバ** *VM* と呼ばれる仮想マシン上に配置され，ホスト間にまたがる IDS VM は仮想ネットワークで接続される．IDS VM 内で動く既存の IDS がサーバ VM を監視できるようにするために，**ソフトウェア・ポートミラーリング**，*VM* **間ディスクマウント**，*VM* **間プロセスマッピング**の 3 つの VM 間監視機構が提供されている．HyperSpector は攻撃者が IDS に直接行う能動的な攻撃を防ぐことができ，IDS が攻撃用コードを含んだデータを読むまで待つ受動的な攻撃の影響を限定することができる．

A Secure Intrusion Detection Architecture Using Virtual Distributed Monitoring Environments

Kenichi Kourai[†] and Shigeru Chiba[†]

We propose a virtual distributed monitoring environment called *HyperSpector*, which achieves secure intrusion detection in distributed systems. While distributed intrusion detection systems (IDSes) can protect servers within a distributed system, their vulnerabilities can increase the number of insecure points in the whole system. HyperSpector overcomes this problem by using virtualization to isolate a distributed IDS from the servers it monitors. The IDSes and servers are located in virtual machines called an *IDS VM* and a *server VM*, respectively, and the IDS VMs among different hosts are connected using a virtual network. To enable legacy IDSes running in the IDS VM to monitor the server VM, HyperSpector provides three inter-VM monitoring mechanisms: *software port mirroring*, *inter-VM disk mounting*, and *inter-VM process mapping*. Consequently, active attacks, which directly attack the IDSes, are prevented. The impact of passive attacks, which wait until data including malicious code is read by an IDS, is confined.

1. はじめに

分散システムでは 1 つのホストがワームやウィルスの攻撃を受けると，分散システム全体にすぐに広まってしまう．できるだけ早くこれらの広がりを防ぐには，分散化された侵入検知システム（IDS）が必要である．分散 IDS [19] は分散システムを守るために各ホストに IDS を配置する．それぞれの IDS が収集した情報は攻撃を解析するためのホストに集められ，その結果は IDS 間で共有される．

分散 IDS は監視している分散システム全体の安全性を低下させてはならない．しかし，現状では分散 IDS 自体の脆弱性が攻撃を受ける機会を増加させてしまっている．IDS に対する攻撃には能動的なものと受動的なものとがある．能動的な攻撃は，攻撃者が IDS を攻撃するために直接行う攻撃である．受動的な攻撃は，攻撃者が IDS の監視対象となるデータに攻撃用コードを挿入しておき，IDS がそのデータの処理を誤ることによって攻撃用コードを実行してしまうのを待つ攻撃である．データを監視することが IDS の役目なので，受動的な攻撃を防ぐのは難しい．そのため，安全な侵入検知には，(1) IDS を能動的な攻撃から守ること，および，(2) IDS に対する受動的な攻撃が分散システムに影響を及ぼさないようにすることが必要とされる．

我々の提案する仮想的な分散監視環境 *HyperSpector* は，仮想マシン（VM）と仮想ネットワークを用いて分散 IDS を監視対象のサーバから分離することで安全な侵入検知を実現する．IDS とサーバはそれぞれ *IDS*

† 東京工業大学情報理工学研究科数理・計算科学専攻
Department of Mathematical and Computing Sciences, Tokyo Institute of Technology

VM，**サーバ** *VM* と呼ばれる VM に配置され，ホスト間にまたがる IDS VM は独立した仮想ネットワークで接続される．IDS VM 内で動作する既存の IDS がサーバ VM を安全に監視できるように，HyperSpector は**ソフトウェア・ポートミラーリング**，*VM* **間ディスクマウント**，*VM* **間プロセスマッピング**の 3 つの VM 間監視機構を提供する．ソフトウェア・ポートミラーリングはサーバ VM が送受信するパケットを IDS VM から取り込むことを可能にする．VM 間ディスクマウントはサーバ VM のファイルシステムを IDS VM に仮想的にマウントし，ファイルシステムの検査を可能にする．また，VM 間プロセスマッピングは IDS VM がサーバ VM 内のプロセスの挙動を調べることを可能にする．

HyperSpector は分散 IDS を能動的な攻撃から守ることができる．サーバ VM はサーバが攻撃を受けた場合でも攻撃者が IDS VM を攻撃するのを防ぎ，仮想ネットワークは外部の攻撃者が不正なパケットを IDS VM に送るのを防ぐ．一方，HyperSpector は受動的な攻撃の影響を 1 つの HyperSpector 環境の中に閉じ込めることができる．IDS VM はその中の攻撃者がサーバ VM に干渉するのを防ぎ，仮想ネットワークは IDS VM 内の攻撃者が HyperSpector 環境の外に不正なパケットを送るのを防ぐ．分散 IDS ごとに HyperSpector 環境を作成すれば，受動的な攻撃を受けた分散 IDS が他の分散 IDS に影響を及ぼさないようにすることもできる．さらに，VM 間監視機構は IDS VM がサーバ VM を監視することだけを許しているため，システムの安全性を低下させることはない．

以下，2 章では IDS への攻撃と安全な侵入検知のための従来のアーキテクチャについて述べる．3 章では HyperSpector と VM 間監視機構について述べる．4 章では HyperSpector の実装の詳細について述べる．5 章では HyperSpector が提供する安全性を確認し，そのオーバヘッドを測定する実験について述べる．6 章で関連研究について述べ，7 章で本稿をまとめる．

2. 安全な侵入検知への障害

分散 IDS による安全な侵入検知を実現するには，分散 IDS を構成するそれぞれの IDS が攻撃に耐えられなければならない．もし 1 つの IDS への攻撃に成功されると，分散システムへの攻撃を検出できなくなる可能性が高まるからである．

2.1 IDS への攻撃

IDS への攻撃は能動的なものと受動的なものに分類される．能動的な攻撃は攻撃者が IDS に直接行う攻撃である．IDS は，たとえ脆弱性を持っていなかったとしても，同一ホスト上で攻撃を受けたサーバを経由して攻撃を受ける可能性がある．サーバは不特定多数のユーザからのリクエストを受け付けるため，サーバへの攻撃の方が成功させやすいためである．攻撃者は IDS プロセスを停止させることで IDS を停止させることができ，IDS が使うポリシファイルを書き換えることで IDS の機能を無効化することもできる．さらに，IDS はネットワーク経由でその脆弱性を利用した攻撃を受ける可能性もある．分散 IDS では IDS どうしが連携するためにログを他の IDS に送信するなどの通信を行う．Counterpane [3] のように攻撃の解析を外部で行う場合には，IDS 間の通信に用いるネットワークポートが攻撃されうる．実際，Big Brother [17] には外部の攻撃者が任意のファイルを読める脆弱性があった[4]．たとえ分散システム外部からは IDS を守ることができたとしても，サーバへの攻撃が成功するとそのサーバ経由で IDS のネットワークポートが攻撃されうる．

他方，受動的な攻撃は攻撃者が IDS の脆弱性を利用する攻撃データを作成し，IDS がそのデータを解析する段階で攻撃に成功するのを待つというものである．典型的な攻撃例は IDS の解析ルーチンにおけるバッファオーバフローの悪用である．ネットワークベース IDS（NIDS）に対しては，攻撃者はサーバに攻撃用のパケットを送り，IDS がそのパケットを解析することで攻撃を成功させることができる[2]．ホストベース IDS（HIDS）に対しては，攻撃者はファイルの中身や属性を書き換えて，IDS がファイルシステムの整合性を検査するためにファイルを読むことで攻撃を成功させることができる[5]．IDS はサーバを監視するという性質上，外部からのデータを読み込まなければならないため，受動的な攻撃を防ぐのは難しい．そのため，攻撃の影響を限定することが重要になる．

2.2 既存の分散監視アーキテクチャの問題

このような IDS に対する攻撃に対処するために，IDS とサーバを分離するアーキテクチャが用いられている．NIDS からなる分散 IDS は**図 1** のように構成することができる．それぞれの NIDS は監視しているサーバとは別のホストに配置される．フロントエンド・スイッチはポートミラーリング機能を持ち，サーバホストは通常のポートに，NIDS ホストはミラーリングポートに接続される．ポートミラーリング機能により，サーバが送受信するパケットは NIDS ホストに転送される．さらに，NIDS ホストは別の NIC を備えており，他の NIDS と通信するためにバックエンド・

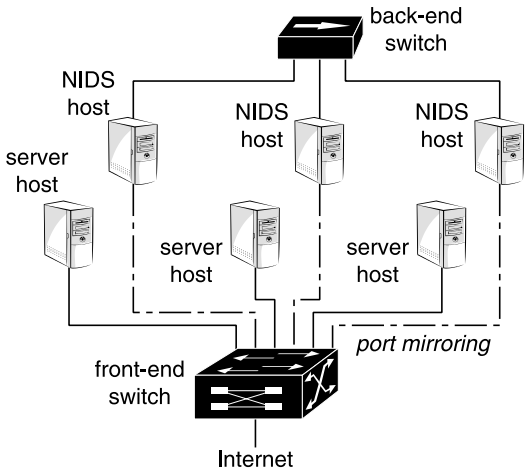

図 1 NIDS 用の分散監視アーキテクチャ
Fig. 1 Isolated monitoring architecture for NIDSes.

スイッチに接続される．

このアーキテクチャは攻撃を受けたサーバ経由での NIDS への能動的な攻撃を防ぐことができる．攻撃を受けたサーバは NIDS ホストのファイルシステムやプロセスに直接アクセスすることはできない．フロントエンド・スイッチのミラーリングポートは監視のためだけにしか使えないので，攻撃者はミラーリングポート経由で NIDS ホストを攻撃することができない．さらに，NIDS ホストは他の NIDS ホストとしか通信できないため，攻撃者は NIDS 用のネットワークにパケットを送って NIDS ホストを攻撃することもできない．また，このアーキテクチャは NIDS への受動的な攻撃の影響を分散 IDS 内部に閉じ込めることができる．NIDS ホストへの受動的な攻撃が成功したとしても，フロントエンド・スイッチのミラーリングポート経由で外部と通信することはできないので，サーバホストや外部ホストを攻撃することはできない．

このアーキテクチャの問題は，NIDS ホストやスイッチなどのハードウェアが増大することである．ポートミラーリング機能を持った高価なフロントエンド・スイッチが必要になるうえ，1 つの NIDS が 1 つのサーバホストを監視する場合，スイッチの半分のポートはミラーリングポートとして使われてしまう．安価なダムハブを用いることも可能ではあるが，バスが共有されるため性能ボトルネックになる．また，ハードウェアが増大することにより管理コストも増大する．

一方，HIDS ホストとサーバホストを分離するには，Backdoor アーキテクチャ[1] を用いることができる．Backdoor 用に HIDS を開発しなければならないが，HIDS からなる分散 IDS も図 1 と同様に構成することができる．Backdoor はポートミラーリング機能を必要としないが，OS を介さずに監視するためにサーバホストにプログラム可能なネットワークインタフェースカード（NIC）を必要とする．また，Backdoor の HIDS ホストはサーバホストから監視データを受信できるようにフロントエンド・スイッチの通常のポートに接続されるため，攻撃者は不正なパケットを HIDS ホストに送ることができる．逆に，HIDS ホストが攻撃を受けるとサーバホストを攻撃することもできる．そのうえ，HIDS ホストからの監視を可能にするためにサーバホストが公開している監視用インタフェースは攻撃対象となりうる．性能面でも，HIDS がサーバホストのすべてのファイルを検査しようとすると，大量のデータをネットワーク経由で取得しなければならないという問題がある．

また，IDS とサーバを分離するために，サーバを VM 上に配置するアーキテクチャも提案されている．ReVirt[7] は IDS が VM の実行をハードウェアレベルで記録することを可能にしている．Livewire[10] は IDS が VM の状態に加えて，VM 内の OS の状態を監視することも可能にしている．SODA[11] は NIDS を VM 上の OS 内で動かし，ログデータを VM の外で記録することができる．これらのアーキテクチャでは，攻撃を受けたサーバ経由での IDS への能動的な攻撃は VM によって防がれる．しかし，IDS への攻撃は想定されていないため，IDS 間で通信を行う分散 IDS を使うと，ネットワーク経由で能動的な攻撃を受ける可能性がある．さらに，IDS が能動的または受動的な攻撃を受けた場合，攻撃者はサーバが動いている VM や他のホストを攻撃することができる．

3. HyperSpector

従来の分散監視アーキテクチャの問題を解決するために，我々は *HyperSpector* と呼ばれる仮想的な分散監視環境を提案する．HyperSpector は仮想化技術を用いることで，ハードウェアを用いて実現されていた分散監視アーキテクチャを追加ハードウェアなしで実現する．さらに，HyperSpector は安全な監視機構を提供することで，既存の HIDS に対しても安全な侵入検知を実現する．

3.1 アーキテクチャ

図 2 に示されるように，HyperSpector はサーバと IDS をそれぞれの VM で動作させる．1 つの IDS が他の IDS に影響を及ぼさないようにするために，それぞれの IDS を異なる VM で動作させることもできる．IDS を動作させる VM は *IDS VM* と呼ばれ，サーバ

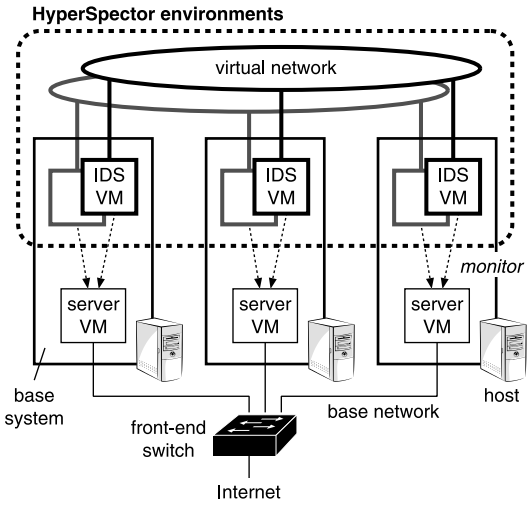

図 2 HyperSpector アーキテクチャ
Fig. 2 HyperSpector architecture.

を動作させる VM は**サーバ** *VM* と呼ばれる．それぞれの VM はネットワークを利用しないベースシステム上で動作し，他の VM やベースシステムから分離されている．IDS VM は物理的なネットワークを使わずに同じホスト上のサーバ VM を監視することができる．ハードウェアベースの分散監視アーキテクチャとは異なり，フロントエンド・スイッチに IDS のためのポートは必要なく，ポートミラーリング機能を備えた高価なスイッチも必要としない．IDS がサーバ VM の大量のデータを監視する必要がある場合でも，フロントエンド・スイッチを経由せずに効率良くサーバ VM から IDS VM にデータを転送できる．

各ホスト上の IDS VM は仮想ネットワークを用いて接続される．仮想ネットワークは LAN などのベースネットワーク上に構築され，追加ハードウェアを必要としない．仮想ネットワークには IDS VM だけが接続され，サーバ VM やベースシステムからは利用できない．逆に，IDS VM は仮想ネットワークだけに接続され，ベースネットワークを利用することはできない．また，仮想ネットワークは管理者によってベースシステムから設定され，IDS VM の中から仮想ネットワークの設定を変更することはできない．IDS が侵入を検知した場合には，管理者はこの仮想ネットワークに接続されたホスト上のウェブブラウザやメールクライアントを用いて警告を受け取ることができる．

1 つの HyperSpector 環境は各ホスト上の 1 つの IDS VM とそれらを接続する 1 つの仮想ネットワークからなる．HyperSpector 環境は分散 IDS を動かすために使われ，システムの他の部分からは独立している．管理者は分散 IDS どうしを分離するために，複数の HyperSpector 環境を作ることもできる．

3.2 攻撃耐性

HyperSpector は分散 IDS を能動的な攻撃から守ることができる．サーバが攻撃を受けたとしても，サーバ VM は同じホスト上の IDS VM やベースシステムへの攻撃を防ぐ．攻撃者がサーバ VM の管理者権限を奪えたとしても，サーバ VM の外のファイルシステムやプロセスにアクセスすることはできない．また，IDS VM だけが接続されている仮想ネットワークは IDS へのネットワーク経由の攻撃を防ぐ．仮想ネットワークはサーバ VM や HyperSpector 外部のホストからの通信を拒否するため，IDS VM が仮想ネットワークの外から不正なパケットを受け取ることはない．

HyperSpector は IDS への受動的な攻撃の影響を 1 つの HyperSpector 環境の中に閉じ込めることができる．たとえ IDS VM 内の IDS が受動的な攻撃を受けたとしても，IDS VM は攻撃者が同じホスト上のサーバ VM やベースシステムのファイルシステムやプロセスに干渉するのを防ぐ．一方，IDS VM は仮想ネットワークを用いて他のホスト上の IDS VM と接続されているため，攻撃者は同一の HyperSpector 環境の中の IDS VM を攻撃することはできる．しかし，攻撃者はその HyperSpector 環境の外側を攻撃することはできない．攻撃者はその HyperSpector 環境の仮想ネットワークを使わずにパケットを送ることができず，仮想ネットワークに他のホストを接続することもできないからである．

3.3 VM 間監視機構

HyperSpector は VM と仮想ネットワークを用いて分散 IDS とサーバを完全に分離する．IDS VM 内で動作する既存の IDS がサーバ VM を監視できるようにするには，既存の IDS へのインタフェースおよび VM 間での安全な監視を提供する機構が必要となる．この 2 つの要求を満たすために，HyperSpector は仮想マシンモニタ（VMM）内で 3 つの VM 間監視機構を提供する．

3.3.1 ソフトウェア・ポートミラーリング

図 3 のように，サーバ VM のネットワーク，IDS VM のネットワーク，ホスト外部のネットワークは VMM 内部の**仮想スイッチ**に接続される．仮想スイッチが外部ネットワークからパケットを受け取ると，そのパケットは適切な VM に配送される．その一方で，仮想スイッチは VM 間ではパケットを配送しない．

仮想スイッチのソフトウェア・ポートミラーリング機能はサーバ VM が送受信するすべてのパケットを IDS

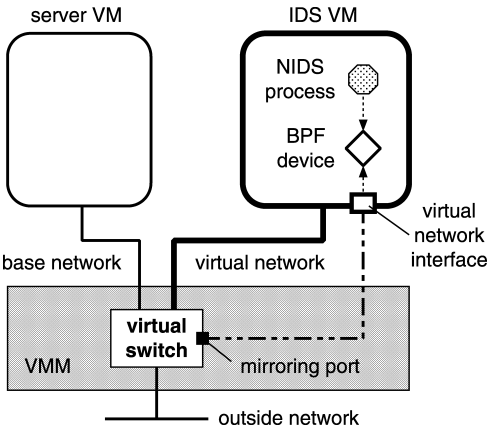

図 3 ソフトウェア・ポートミラーリング機能付きの仮想スイッチを用いたネットワークの監視
Fig. 3 Network monitoring using the virtual switch with the software port mirroring feature.

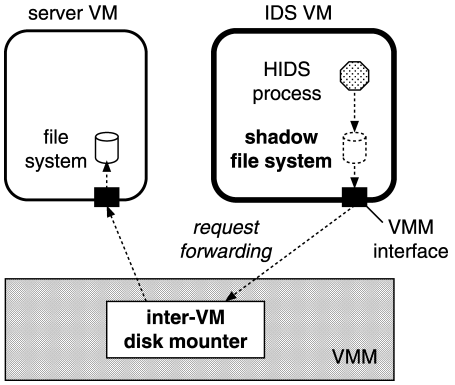

図 4 VM 間ディスクマウンタを用いたファイルシステムの監視
Fig. 4 File system monitoring using the inter-VM disk mounter.

VM が取り込むことを可能にする．IDS VM は仮想スイッチの通常のポートに加えて，**仮想ネットワークインタフェース**を介してミラーリングポートにも接続される．サーバ VM が送受信するすべてのパケットは仮想スイッチで複製され，ミラーリングポートに転送される．そのパケットは IDS VM の仮想ネットワークインタフェースに転送され，IDS VM は Berkeley パケットフィルタ（BPF）などの既存のパケットフィルタデバイスを使ってパケットを取り込むことができる．

ソフトウェア・ポートミラーリングは IDS VM がサーバ VM を監視するためだけに使うことができる．仮想スイッチは IDS VM が送受信するパケットをミラーリングポートに複製しないため，サーバ VM が IDS VM のパケットを取り込むことはできない．また，仮想スイッチのミラーリングポートはパケットを転送するためだけに使われ，ミラーリングポートに送られたパケットは破棄される．そのため，IDS VM が受動的な攻撃を受けても，攻撃者はミラーリングポートを介してサーバ VM や外部ホストを攻撃することはできない．一方，攻撃を受けた IDS VM の中の攻撃者はこの監視機構を使ってサーバ VM の機密情報を見ることができる．しかし，攻撃者が機密情報を盗めたとしても，その情報を HyperSpector 環境の外に持ち出すことはできない．

3.3.2 VM 間ディスクマウント

VM 間ディスクマウントはサーバ VM のファイルシステムの検査を可能にする．サーバ VM が使うディスクは IDS VM からアクセスできないので，IDS VM はサーバ VM のディスクを直接マウントすることはできない．IDS VM はサーバ VM と直接通信できないので，NFS を使ってサーバ VM のファイルシステムをマウントすることもできない．そこで，我々は *VM* **間ディスクマウンタ**を開発し，サーバ VM のファイルシステムが**シャドーファイルシステム**として IDS VM 上に仮想的にマウントされるようにした．

VM 間ディスクマウンタは IDS VM のシャドーファイルシステムと監視対象のサーバ VM のファイルシステムを**図 4** のように仲介する．IDS が既存のシステムコール経由でシャドーファイルシステムにアクセスしたとき，IDS VM は *VMM* **インタフェース**を用いてアクセス要求を VM 間ディスクマウンタに転送する．VM 間ディスクマウンタはその要求をサーバ VM の VMM インタフェースに転送し，サーバ VM はその要求を処理して IDS VM に結果を返す．

VM 間ディスクマウントは IDS VM だけにサーバ VM のファイルシステムを監視することを許可する．サーバ VM から IDS VM へのマウント要求は VM 間ディスクマウンタによって拒否される．ファイルの書き込みなどファイルシステムへの変更をともなう要求は IDS VM が発行したとしても拒否される．そのため，攻撃を受けた IDS VM 内の攻撃者はシャドーファイルシステムを経由してサーバ VM のファイルシステムを書き換えることはできない．

3.3.3 VM 間プロセスマッピング

VM 間プロセスマッピングは IDS VM がサーバ VM のプロセスの状態を検査することを可能にする．IDS VM とサーバ VM は異なるプロセス空間を持つため，IDS VM は直接サーバ VM のプロセスを監視することはできない．そこで，我々は *VM* **間プロセスマッパ**を開発し，サーバ VM の全プロセスが**シャドープロセス**として IDS VM に自動的にマップされるようにした．

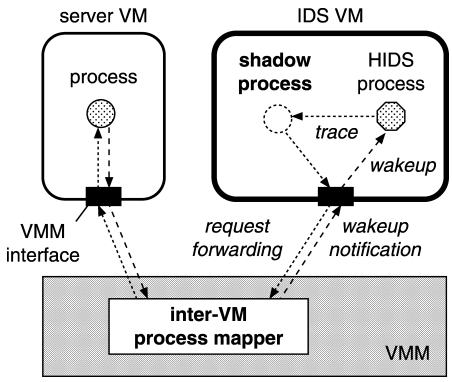

図 5 VM 間プロセスマッパを用いたプロセスの監視
Fig. 5 Process monitoring using the inter-VM process mapper.

VM 間プロセスマッパは IDS VM のシャドープロセスと監視対象のサーバ VM のプロセスとを**図 5** のように仲介する．たとえば，IDS が既存の `ptrace` システムコールや proc ファイルシステム経由でシャドープロセスに追跡要求を送ったとき，IDS VM は VMM インタフェースを用いてその要求を VM 間プロセスマッパに転送する．VM 間プロセスマッパは要求に含まれるプロセス識別子をサーバ VM の対応する識別子に変換し，変換された要求をサーバ VM の VMM インタフェースに送る．サーバ VM はその要求を処理した後，結果を IDS VM に返す．一方，追跡対象のプロセスがシステムコールを発行したとき，そのイベントを追跡元の IDS プロセスに通知する必要がある．そのために，wakeup 通知がサーバ VM の VMM インタフェースに送られ，VM 間プロセスマッパを経由して IDS VM に転送される．IDS VM は追跡元の IDS プロセスを探し，そのプロセスを起こす．

VM 間プロセスマッピングは IDS VM がサーバ VM のプロセスを監視することだけを許可する．シャドープロセスはサーバ VM には作られず，VM 間プロセスマッパに登録されていないプロセス識別子へのアクセス要求は拒否される．IDS VM はシャドープロセスのレジスタとメモリを読むことだけを許されており，攻撃を受けた IDS VM 内の攻撃者はサーバ VM のプロセスのメモリを書き換えることはできない．

4. 実　　装

我々は FreeBSD 4.9 をベースにした Persona OS に HyperSpector を実装した．HyperSpector の各ホストはベースシステム，IDS VM，サーバ VM，VMM からなる．我々の実装では，カーネルおよびベースシステムは攻撃を受けないと仮定している．この章ではまず，IDS VM とサーバ VM の実装について述べ，次に VMM における VM 間監視機構の実装の詳細について述べる．

4.1 IDS VM

IDS VM は我々が以前に開発したポートスペース[14)]を拡張して実装した．ポートスペースはハードウェアをエミュレートする VM ではなく，独立したファイルシステム，ネットワークシステム，プロセスの名前空間だけを提供する．我々はカーネルは攻撃を受けないように注意深く開発されていると仮定しているので，このような仮想化で IDS VM は十分安全になる．この章では拡張されたポートスペースの概要について述べる．ポートスペースの実装の詳細については文献 14) で述べられている．

4.1.1 ネットワークシステムの仮想化

仮想ネットワークは IPsec トンネルを使って実装されている．特定の IDS VM だけが特定の IPsec トンネルを使えるように，IDS VM はトンネルインタフェース，IPsec データベース，経路表，プロトコル制御ブロックを含む独立したネットワークシステムを提供する．IDS VM が攻撃を受けたときでも管理者による設定を強制できるように，仮想ネットワークはベースシステムからポートスペースを指定する `setportspace` システムコールを発行した後で設定される．

4.1.2 ファイルシステムの仮想化

Persona は我々が改良した union ファイルシステムを用いて，IDS VM ごとに `/.filespace/<id>`（`<id>` は IDS VM の ID）を `/` の上に透過的に重ね合わせてマウントする．これにより，IDS VM は union ファイルシステムの下位層で提供される標準的なバイナリやライブラリを利用しつつ，上位層で提供される専用のファイルシステムも利用できる．IDS VM 内のプロセスはつねに上位層（`/.filespace/<id>`）にファイルを書き込むので，下位層（ベースシステムのファイルシステム）に悪影響を及ぼすことはない．一方，IDS VM でファイルが上書きされない限り，IDS VM 内のプロセスは下位層からファイルを読み込む．我々はベースシステムは攻撃を受けないと仮定しているので，下位層からのファイルの読み込みが IDS VM に対する受動的な攻撃を引き起こすことはない．

4.1.3 プロセスの仮想化

IDS VM 内のプロセスはほぼ FreeBSD の通常のプロセスであるが，IDS VM の外部のプロセスとは異なる名前空間を持つ．このため，IDS VM の内部と外部のプロセスは互いにシグナルや共有メモリなどを用いてプロセス間通信を行うことはできない．

4.2 サーバ VM

サーバ VM もポートスペースをベースにして実装されている．ファイルシステムとプロセスの仮想化は IDS VM と同様に行われているが，サーバ VM はネットワークシステムを仮想化せず，仮想ネットワークも使用しない．その代わり，サーバ VM はサーバがインターネットにサービスを提供できるようにベースネットワークを使用する．

4.3 VMM

カーネル内に実装された VMM は VM 間監視機構を実現するために仮想スイッチ，VM 間ディスクマウンタ，VM 間プロセスマッパを提供している．

4.3.1 仮想スイッチ

仮想スイッチは VM から外部ネットワークへのパケットは直接送信するが，外部ネットワークからのパケットは IPsec ヘッダのセキュリティ・パラメータ・インデックスに基づいて適切な VM に配送される．IPsec ヘッダを持たないパケットはサーバ VM に配送される．仮想スイッチでソフトウェア・ポートミラーリングを実現するために，Persona はサーバ VM のネットワークインタフェースを IDS VM の仮想ネットワークインタフェースにマップする．IDS が仮想ネットワークインタフェースに対して BPF デバイスをオープンすると，仮想スイッチはサーバ VM が送受信するパケットをそのインタフェースにコピーする．性能劣化を防ぐために，IDS が BPF デバイスをオープンしない限りパケットはコピーされない．

4.3.2 VM 間ディスクマウンタ

我々の実装では，VMM の VM 間ディスクマウンタ，IDS VM のシャドーファイルシステム，サーバ VM のファイルシステムは同一のカーネル内に存在するため，VM 間ディスクマウントは効率良く実装されている．IDS VM 内の IDS がシャドーファイルシステムからファイルを読んだとき，ファイルの内容はサーバ VM のディスクからカーネル内のバッファに読み込まれる．このバッファは VM 間で共有されているため，IDS VM はコピーなしでファイルの内容を得ることができる．オーバヘッドは IDS VM の `/.serverfs` にマウントされるシャドーファイルシステムを介してサーバ VM のファイルシステムに間接的にアクセスすることでのみ発生する．

VM 間ディスクマウンタはサーバ VM で使われている union ファイルシステム全体だけでなく，その上位層部分（変更部分）も IDS VM にマウントする．この上位層部分はシャドーファイルシステムを介して IDS VM の `/.dserverfs` にマウントされ，IDS VM がサーバ VM のファイルシステムの変更部分だけを効率良く監視することを可能にする．変更部分のみで十分な監視を行うには union ファイルシステムに関する詳細な情報が必要となるため，Persona は `ustat` システムコールを提供している．このシステムコールは下位層フラグ（`SF_LOWER`）とホワイトアウトモード（`S_IFWHT`）を返せるように `lstat` システムコールを拡張したものである．`SF_LOWER` はファイルが union ファイルシステムの下位層にあることを意味する．`S_IFWHT` は下位層に実体のある union ファイルシステム上のファイルが削除されたことを意味する．これらの付加情報を用いることで，IDS は**表 1** のように，現在の `/.dserverfs` と変更前の `/.serverfs` の情報を比較してファイルへの変更を判断することができる．

表 1 `/.dserverfs` を用いたファイルへの変更の判定表（`!` はその属性がついていないことを表す）
Table 1 Table for changes to a file by using `/.dserverfs`.

`/.dserverfs`	変更前の `/.serverfs`			
	存在せず	`S_IFWHT`	`!S_IFWHT`	
			`SF_LOWER`	`!SF_LOWER`
存在せず	不変	–	不変	削除
`S_IFWHT`	–	不変	削除	–
存在	追加	追加	変更	変更の疑い

4.3.3 VM 間プロセスマッパ

同様に，VMM の VM 間プロセスマッパ，IDS VM のシャドープロセス，サーバ VM のプロセスも同一カーネル内に存在するため，VM 間プロセスマッピングも効率良く実装されている．プロセス情報はサーバ VM と IDS VM とで共有されているため，IDS プロセスは VM 間でのコピーなしにシャドープロセスを介して監視対象のプロセスのレジスタやメモリの内容を得ることができる．さらに，シャドープロセスを除くすべてのプロセスはホスト内で一意の識別子を持ち，IDS VM 内のシャドープロセスにはサーバ VM 内の対応するプロセスと同一の識別子が割り当てられている．そのため，VM 間プロセスマッパはプロセス識別子を変換する必要がない．

現在の実装では，IDS はシャドープロセスの識別子を直接指定することによってのみ監視を行うことができる．これは，IDS からサーバプログラムを子プロセスとして実行させると，IDS プロセスとサーバプロセスが同じ VM で動くことになり，IDS とサーバの分離ができなくなるためである．

5. 実　　験

我々は HyperSpector の安全性の確認およびオーバ

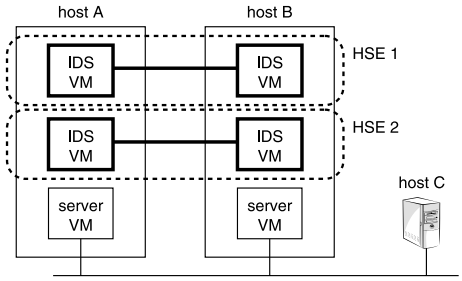

図 6 実験に用いたシステム設定
Fig. 6 System configuration for experiments.

ヘッド測定のための実験を行った．この実験では**図 6** のように 2 つの HyperSpector 環境（HSE 1 および HSE 2）を作成した．ホスト A およびホスト C にはそれぞれ 3.0 GHz の Pentium 4 を持つ PC を用い，ホスト B には 2.8 GHz の Pentium 4 を持つ PC を用いた．これら 3 台の PC は 1 GB のメモリ，Intel Pro/100+の NIC を備え，100Base-T のスイッチで接続された．

5.1 安 全 性

HyperSpector の安全性を確認するために，我々は IDS への能動的な攻撃と受動的な攻撃の影響を調べた．この実験では，サーバ VM で httpd を実行し，IDS VM で ftp サービスを有効にした inetd を実行した．

5.1.1 ネットワークの安全性

ネットワーク経由の攻撃への耐性を調べるため，nmap [9] を使ってポートスキャンを行った．第 1 に，外部からの能動的な攻撃として，ホスト C から HSE 1 の IDS VM にポートスキャンを行った．第 2 に，攻撃を受けたサーバ経由での能動的な攻撃として，サーバ VM から HSE 1 の IDS VM にポートスキャンを行った．第 3 に，受動的な攻撃を受けた IDS VM 経由の攻撃として，HSE 1 の IDS VM からホスト C，サーバ VM，HSE 1 内の別の IDS VM にポートスキャンを行った．最後に，HyperSpector 環境間の攻撃として，HSE 2 の IDS VM から HSE 1 の IDS VM にポートスキャンを行った．

表 2 はこれらのポートスキャンの結果を示している．ホスト C とサーバ VM からのポートスキャンの結果より，IDS VM は能動的な攻撃を受けないことが確認できた．一方，IDS VM 1 と IDS VM 2 からのポートスキャンの結果より，受動的な攻撃を受けた IDS VM 上の攻撃者は同じ HyperSpector 環境内の IDS VM 以外を攻撃することはできないことが確認できた．

次に，ソフトウェア・ポートミラーリングの安全性を調べるために，この監視機構の不正利用を試みた．IDS VM からベースネットワークにパケットを送れるように，`route` コマンドを使って仮想ネットワークインタフェースを指定した経路の追加を試みたが，この設定は拒否された．一方，サーバ VM には IDS VM のネットワークインタフェースがマップされておらず，パケットの取り込みも経路の追加もできなかった．これらの結果より，IDS VM によるパケットの取り込み以外は許されないことが確認できた．

表 2 ポートスキャンの結果（IDS VM 1 は HSE 1 内の IDS VM を意味する）
Table 2 Results of port scanning.

スキャン元	スキャン対象		
	ホスト C	サーバ VM	IDS VM 1
ホスト C	–	–	未到達
サーバ VM	–	–	未到達
IDS VM 1	未到達	未到達	ftp
IDS VM 2	–	–	未到達

5.1.2 ファイルシステムの安全性

VM 間ディスクマウントを用いない場合には，サーバ VM と IDS VM のそれぞれのファイルシステムの全ファイルを削除しても，他の VM およびベースシステムのファイルシステムには影響がなかった．次に，VM 間ディスクマウントの安全性を調べるために，この監視機構の不正利用を試みた．IDS VM からシャドーファイルシステムがマウントされた `/.serverfs` の全ファイルの削除を試みたが，どのファイルも削除できなかった．一方，サーバ VM には IDS VM のシャドーファイルシステムが存在せず，アクセスできなかった．これらの結果から，IDS VM によるファイルシステムの検査以外は許されないことが確認できた．

5.1.3 プロセスの安全性

VM 間プロセスマッピングを用いない場合は，サーバ VM と IDS VM で ps コマンドを実行してもそれぞれで実行されている httpd と inetd しか見えず，互いに干渉できなかった．次に，VM 間プロセスマッピングの安全性を調べるために，この監視機構の不正利用を試みた．まず，IDS VM から `kill` システムコールを用いて，シャドープロセス経由でサーバ VM の httpd プロセスにシグナルを送ってみた．さらに，`ptrace` システムコールを用いて httpd プロセスの変更も試みたが，これらの試みは拒否された．一方，サーバ VM には IDS VM で実行されている inetd のシャドープロセスが存在せず，アクセスできなかった．これらの結果より，IDS VM によるプロセスの監視以外は許されないことが確認できた．

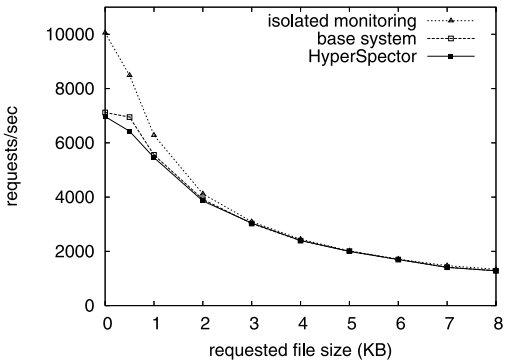

図 7 Snort によって監視された thttpd の性能
Fig. 7 Performance of thttpd with Snort.

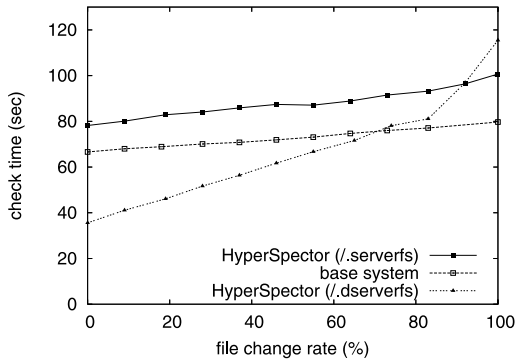

図 8 Tripwire における整合性検査にかかる時間
Fig. 8 Time for integrity checking in Tripwire.

5.2 オーバヘッド

HyperSpector は IDS VM からサーバ VM を監視することによるオーバヘッドを被る．我々は既存の IDS についてこのオーバヘッドを測定した．

5.2.1 Snort

Snort [18] はパケットのパターンマッチで攻撃を検出する NIDS である．我々は Snort に監視されているホスト上で動く thttpd 2.23 beta1 ウェブサーバ[16] の性能を測定した☆．thttpd と Snort は，(1) ホスト A のベースシステム上，(2) HyperSpector を用いてホスト A のサーバ VM と IDS VM 上，(3) 従来の分散監視アーキテクチャを用いてホスト A とホスト B 上に配置された．これら 3 つの構成に対してホスト C でリクエストされるファイルサイズを変えて ApacheBench ベンチマークを実行したところ，thttpd の性能は**図 7** のようになった．ベースシステムと比較すると HyperSpector のオーバヘッドは最大で 7.5%であり，リクエストされるファイルサイズが大きくなるにつれてオーバヘッドは減少した．一方，従来の分散監視アーキテクチャと比較すると，リクエストされるファイルサイズが 0 のときは性能が 30%低下した．これは分散監視アーキテクチャでは Snort のオーバヘッドは thttpd には影響しないためである．しかし，リクエストするファイルサイズが 2 KB を超えると，HyperSpector のオーバヘッドは 7%以下になった．

5.2.2 Tripwire

Tripwire [13] はファイルシステムの整合性を検査する HIDS である．我々はホスト A の IDS VM で実行される Tripwire がサーバ VM のファイルシステムを検査するのにかかる時間を測定した．比較のため，ベースシステムで Tripwire を実行した場合についても測定した．また，表 1 に基づいて，サーバ VM のファイルシステムの変更部分だけを検査するように改良した Tripwire を IDS VM で実行した場合についても測定した．実験には Tripwire 2.3.1 を用い，54,885 個のファイルが検査された．検査の前後で変更されるファイル数の割合を 0〜100%まで変えて測定を行ったところ，結果は**図 8** のようになった．この結果から，HyperSpector のオーバヘッドは 17〜26%であり，変更したファイルの割合が大きくなるにつれてオーバヘッドも大きくなることが分かった．一方，ファイルシステムの変更部分だけを検査する Tripwire を用いた場合，ファイルの変更率が 70%を超えるまでは検査にかかる時間がベースシステムでの時間よりも短くなった．この結果より，ファイルがそれほど変更されない一般的な状況では大幅に性能を改善できていることが分かる．

5.2.3 Truss

truss はシステムコールを追跡する FreeBSD 標準コマンドである．truss は IDS ではないが，VM 間でシステムコールを追跡するオーバヘッドを見積もるために，truss によって追跡される thttpd の性能を測定した．この実験では thttpd ウェブサーバと truss をそれぞれホスト A 上のサーバ VM と IDS VM で実行した．比較のために，thttpd と truss をベースシステムで実行したときの性能も測定した．ホスト C でリクエストされるファイルサイズを変えて ApacheBench を実行したところ，thttpd の性能は**図 9** のようになった．この結果から，VM 間でシステムコールを追跡するオーバヘッドは 0.8〜7.3%であることが分かった．

6. 関 連 研 究

IDS のアーキテクチャに関する関連研究は 2.2 節で

☆ 複数のプロセスを起動する Apache ウェブサーバを用いると高負荷時に Snort が十分に CPU を獲得できずパケットを取りこぼすため，プロセスを 1 つだけ起動する thttpd を用いた．

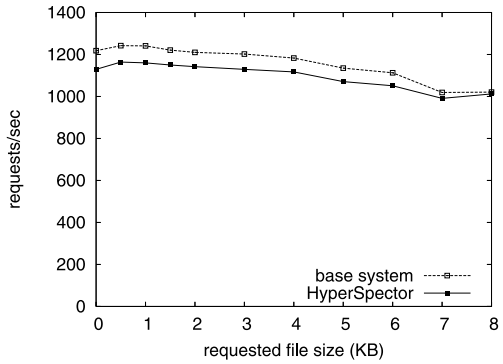

図 9 truss によって追跡される thttpd の性能
Fig. 9 Performance of thttpd traced by truss.

あげた．ここでは我々が利用した仮想化技術に関する関連研究について述べる．

VM と仮想ネットワークの組合せが安全な分散コンピューティングを実現するために使われている．Figueiredo らは VM と仮想ネットワークを用いて Grid ユーザ同士の実行環境を分離するアーキテクチャを提案している[8]．このアーキテクチャは信頼できないプログラムを Grid 上で安全に実行することを可能にするが，実行を監視する機構は提供されていない．我々の以前の研究であるパーソナルネットワーク[14),22)]は，ユーザが複数のネットワークを安全に使い分けられるようにするために VM と仮想ネットワークを用いている．パーソナルネットワークは監視機構を提供しておらず，ユーザが仮想ネットワークを自由に拡張できる．

IDS VM やサーバ VM を実装するために我々はポートスペースを拡張したが，すでに開発されている VM を拡張することも可能である．FreeBSD の jail[12)] は独立したファイルシステムとプロセス空間を提供し，異なる IP アドレスを割り当てられる．Cloneable Network Stacks[21)] は独立したネットワークスタックおよびファイルシステムを提供する．Zap[15)] はプロセスに仮想化された OS の見え方を提供する．これらはポートスペースと同様に軽量な VM である．このような VM を用いて HyperSpector を実現するには，OS カーネル内に VM 間監視機構を実装する必要がある．他方，VMware[20)] や UML[6)] では VM 内で異なる OS（ゲスト OS）を動かすことができる．このような VM を用いて HyperSpector を実現するには，VMM およびゲスト OS を拡張して VM 間監視機構を実装する必要がある．

7. Conclusion

本稿では，HyperSpector と呼ばれる仮想的な分散監視環境を提案した．HyperSpector は追加ハードウェアなしで分散 IDS を監視対象のサーバから分離することができる．そのために，IDS とサーバはそれぞれ IDS VM とサーバ VM に配置され，各ホスト上の IDS VM は仮想ネットワークに接続される．IDS VM はソフトウェア・ポートミラーリング，VM 間ディスクマウント，VM 間プロセスマッピングの 3 つの VM 間監視機構を用いてサーバ VM を監視する．このように仮想化技術を用いることで，HyperSpector は分散 IDS を能動的な攻撃から守り，受動的な攻撃の影響を限定することができる．

今後の課題としては，HyperSpector をサービス妨害攻撃から守ることがあげられる．攻撃者がサーバに大量のパケットを送ると IDS VM 上の NIDS が過負荷になってしまう．この種の攻撃を防ぐには，HyperSpector 環境が十分な資源を確保できるようにする必要がある．また，受動的な攻撃を受けた HyperSpector 環境を自動的に検出する機構も必要である．攻撃を受けた HyperSpector 環境は正しく侵入検知できなくなるからである．我々は資源の利用状況を監視することで HyperSpector 環境の異常を検出することを検討している．別の課題としては，IDS が侵入を検知した場合にサーバ VM に対して適切な対処を自動的に行えるようにすることがあげられる．IDS VM からサーバ VM を操作できるようにするには，安全性について十分考慮する必要がある．

謝辞 研究に関して適切な助言をくださった千葉研究室の方々，NTT 未来ねっと研究所の方々，および本稿の執筆にあたり有益な助言をくださった査読者の方々に感謝いたします．

参 考 文 献

1) Bohra, A., Neamtiu, I., Gallard, P., Sultan, F. and Iftode, L.: Remote Repair of Operating System State Using Backdoors, *Proc. IEEE Int'l Conf. on Autonomic Computing*, pp.256–263 (2004).
2) CERT: Multiple Vulnerabilities in Snort Preprocessors, CERT Advisory CA-2003-13.
3) Counterpane Internet Security, Inc.: Counterpane. http://www.counterpane.com/
4) CVE: CAN-1999-1462. http://www.cve.mitre.org/
5) CVE: CAN-2004-0536. http://www.cve.mitre.org/
6) Dike, J.: A User-mode Port of the Linux Kernel, *Proc.Linux Showcase & Conference* (2000).
7) Dunlap, G., King, S., Cinar, S., Basrai, M. and

Chen, P.: ReVirt: Enabling Intrusion Analysis through Virtual-machine Logging and Replay, *Proc. Symposium on Operating Systems Design and Implementation*, pp.211–224 (2002).
8) Figueiredo, R., Dinda, P. and Fortes, J.: A Case for Grid Computing on Virtual Machines, *Proc. IEEE Conf. on Distributed Computing Systems*, pp.550–559 (2003).
9) Fyodor: The Network Mapper. http://www.insecure.org/nmap/
10) Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proc. Network and Distributed Systems Security Symposium*, pp.191–206 (2003).
11) Jiang, X., Xu, D. and Eigenmann, R.: Protection Mechanisms for Application Service Hosting Platforms, *Proc. Symposium on Cluster Computing and the Grid* (2004).
12) Kamp, P. and Watson, R.: Jails: Confining the Omnipotent Root, *Proc. Int'l SANE Conf.* (2000).
13) Kim, G. and Spafford, E.: The Design and Implementation of Tripwire: A File System Integrity Checker, *Proc. ACM Conf. on Computer and Communications Security*, pp.18–29 (1994).
14) Kourai, K., Hirotsu, T., Sato, K., Akashi, O., Fukuda, K., Sugawara, T. and Chiba, S.: Secure and Manageable Virtual Private Networks for End-users, *Proc. Annual IEEE Conf. on Local Computer Networks*, pp.385–394 (2003).
15) Osman, S., Subhraveti, D., Su, G. and Nieh, J.: The Design and Implementation of Zap: A System for Migrating Computing Environments, *Proc. Symposium on Operating Systems Design and Implementation*, pp.361–376 (2002).
16) Poskanzer, J.: Tiny/turbo/throttling HTTP Server. http://www.acme.com/software/thttpd/
17) Quest Software: Big Brother systems and network monitor. http://www.quest.com/bigbrother/
18) Roesch, M.: Snort — Lightweight Intrusion Detection for Networks, *Proc. USENIX System Administration Conf.* (1999).
19) Snapp, S., Brentano, J., Dias, G., Goan, T., Grance, T., Heberlein, L., Ho, C., Levitt, K., Mukherjee, B., Mansur, D., Pon, K. and Smaha, S.: A System for Distributed Intrusion Detection, *Proc. COMPCON*, pp.170–176 (1991).
20) VMware, Inc.: VMware. http://www.vmware.org/
21) Zec, M.: Implementing a Clonable Network Stack in the FreeBSD Kernel, *Proc. USENIX Annual Technical Conf.*, pp.137–150 (2003).
22) 光来健一，廣津登志夫，佐藤孝治，明石　修，福田健介，菅原俊治，千葉　滋：VPN とホストの実行環境を統合するパーソナルネットワーク，コンピュータソフトウェア，Vol.21, No.1, pp.2–12 (2004).

(平成 17 年 4 月 28 日受付)
(平成 17 年 6 月 30 日採録)

光来　健一（正会員）

1975 年生．2002 年東京大学大学院理学系研究科情報科学専攻博士課程修了．同年日本電信電話株式会社入社，未来ねっと研究所勤務．2003 年より東京工業大学大学院情報理工学研究科数理・計算科学専攻助手．博士（理学）．オペレーティングシステム，ネットワークの研究に従事．日本ソフトウェア科学会，ACM，IEEE-CS 各会員．

千葉　　滋（正会員）

1968 年生．1991 年東京大学理学部情報科学科卒業．1996 年東京大学大学院理学系研究科情報科学専攻博士課程退学．東京大学助手，筑波大学講師を経て，現在，東京工業大学大学院情報理工学研究科助教授．理学博士．言語処理系およびオペレーティングシステム等システムソフトウェアの研究に従事．日本ソフトウェア科学会，ACM 各会員．