AMD SEVで保護されたVMの隔離エージェントを用いた安全な監視

能野 智玄[1]　光来 健一[1]

概要：クラウドの内部犯による仮想マシン（VM）のメモリの盗聴を防ぐために，VM のメモリデータを透過的に暗号化する AMD SEV が用いられている．一方，VM 内に侵入されると SEV ではメモリ上の機密情報が保護できないため，侵入検知システム（IDS）を用いて VM 内のシステムを監視する必要がある．侵入後に IDS が無力化されるのを防ぐには IDS を VM の外で動作させる IDS オフロードが有効であるが，オフロードされた IDS は SEV を用いて暗号化された VM のメモリから情報を取得することができない．本稿では，SEV を用いてメモリが暗号化された VM の内部でエージェントを安全に動作させることにより IDS オフロードを実現するシステム SEVmonitor を提案する．SEVmonitor はオフロードされた IDS も SEV で保護された別の VM 内で動作させ，エージェントと安全に通信を行うことによって監視対象 VM のメモリデータを取得する．エージェントの安全性を向上させるために，監視対象システムを VM 内の隔離環境に閉じ込め，エージェントをその外側で動作させる．SEVmonitor を KVM と Linux，BitVisor，Xen を用いて実装し，監視対象システムの OS データを取得する性能を調べた．

Secure Monitoring of VMs Protected by AMD SEV with Isolated Agents

Tomoharu Nono[1]　Kenichi Kourai[1]

Abstract: To prevent the memory of virtual machines (VMs) from being eavesdropped on by insiders in clouds, AMD SEV is used for transparently encrypting the memory of VMs. Since SEV cannot protect sensitive information in memory if attackers intrude into VMs, it is necessary to use intrusion detection systems (IDS). IDS offloading is useful to run IDS outside VMs and prevent IDS from being disabled by intruders. However, offloaded IDS cannot obtain information from the memory of VMs encrypted by SEV. In this paper, we propose SEVmonitor for enabling IDS offloading by securely running agents inside VMs. SEVmonitor also runs offloaded IDS in another VM protected by SEV and obtains memory data from the agents in the monitored VMs. To enhance the security of the agents, it confines a monitored system in an isolated environment and runs the agent outsides that environment. We implemented SEVmonitor using KVM, Linux, BitVisor, and Xen, and examined the performance of obtaining OS data.

1. はじめに

近年，ユーザに仮想マシン（VM）を提供する IaaS 型クラウドが普及している．クラウドでより重要なデータを扱うようになるにつれて，クラウドの内部犯からユーザの VM のメモリ上にある機密情報が盗まれるリスクが問題になっている．VM のメモリを保護するために，AMD 製の CPU では SEV [1] と呼ばれる透過的なメモリ暗号機能が提供されている．SEV は VM のメモリを暗号化し，VM 内でアクセスされる時にだけ復号する．SEV を用いることによって，VM を動作させているハイパーバイザでさえ VM のメモリ上にある機密情報にアクセスすることはできなくなる．

一方，VM 内に侵入されてしまうと SEV によるメモリ暗号化による保護は機能しないため，VM 内の機密情報を盗み見られてしまう恐れがある．そのため，侵入検知シ

[1] 九州工業大学
Kyushu Institute of Technology

ステム（IDS）を用いて VM 内のシステムの監視を行うことが必要である．VM への侵入時に IDS が無力化されるのを防ぐために，IDS を VM の外で動作させる IDS オフロード [2] という手法が用いられている．オフロードされた IDS は VM のメモリを解析することによって VM 内のシステムの監視を行う．しかし，SEV を用いて VM のメモリが暗号化されると，IDS オフロードによって VM の監視を行うことができなくなる．

本稿では，SEV を用いてメモリが暗号化された VM に対して IDS オフロードを実現するシステム SEVmonitor を提案する．SEVmonitor は監視対象 VM の内部でエージェントを安全に動作させる．オフロードされた IDS はエージェントから VM のメモリデータを取得することによって，メモリが暗号化された VM の監視を可能にする．IDS 経由での情報漏洩を防ぐために，IDS も SEV によって保護された別の VM 内で動作させる．IDS とエージェントは VM 間の仮想ネットワークまたは共有メモリを用いて通信を行う．通信データは SEVmonitor が独自の暗号化を行い，クラウドの内部犯への情報漏洩を防ぐ．

エージェントの安全性を向上させるために，SEVmonitor は監視対象システムを VM 内の隔離環境に閉じ込め，エージェントをその外側で動作させる．監視対象システムをコンテナに隔離する場合，エージェントは監視対象 VM の中の OS カーネル内に配置する．監視対象システムを内部 VM に隔離する場合，エージェントは監視対象 VM の中のハイパーバイザ内に配置する．我々は SEVmonitor を KVM の VM 内で動作する Linux，BitVisor，Xen に対して実装した．実験により，SEV を用いてメモリが暗号化された VM から OS のバージョン情報とプロセス情報を取得する性能を調べた．

以下，2 章では AMD SEV を用いてメモリが暗号化された VM を監視する際の問題点について述べる．3 章では VM の内部でエージェントを安全に動作させることにより IDS オフロードを実現するシステム SEVmonitor を提案する．4 章では SEVmonitor の実装について説明する．5 章では SEVmonitor を用いて行った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. AMD SEV で保護された VM の監視

クラウド内には悪意のある管理者などの内部犯がいる可能性が指摘されている [3]．内部犯によって VM のメモリが盗聴されると VM 内に格納されているユーザ情報などの機密情報が盗まれる恐れがある．この問題を解決するために，AMD SEV と呼ばれる CPU のセキュリティ機構が提供されており，VM のメモリを透過的に暗号化することができる．SEV はメモリコントローラに組み込まれた暗号化エンジンを用いて VM ごとに異なる暗号鍵でメモリの暗号化を行う．これらの鍵はセキュアプロセッサによって安全に管理されるため，VM の管理を行う権限を持った内部犯であっても VM のメモリを盗聴することはできない．

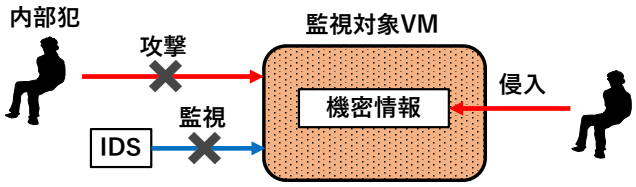


図 1: SEV を用いてメモリが暗号化された VM への攻撃
Fig. 1 Attacks against VMs whose memory is encrypted by SEV.

一方，VM のメモリが SEV によって暗号化されていても，図 1 のようにネットワーク経由で VM 内に侵入されると SEV による保護は機能しない．SEV によるメモリ暗号化は VM 外部からの攻撃に対してのみ有効であるためである．VM 内部ではメモリは CPU によって透過的に復号されるため，VM に侵入した攻撃者はメモリ上の機密情報を容易に盗聴することができる．OS などのソフトウェアの脆弱性を利用したり，マルウェアを動作させたりすることで機密情報を取得することもできる．

そこで，従来と同様に IDS を用いて VM 内部での攻撃を検知する必要がある．VM に侵入された際に IDS が無力化され，それ以降の攻撃が検知できなくなる事態を防ぐために，VM の外で IDS を動作させる IDS オフロードと呼ばれる手法が用いられている [2]．VM の外にオフロードされた IDS は監視対象 VM のメモリから OS データを取得して解析することによって攻撃を検知する．そのため，VM 内に侵入されてもオフロードされた IDS は攻撃を検知し続けることができる．

しかし，SEV を適用した VM に対して IDS オフロードを行うには問題が 2 つある．第一に，VM のメモリが暗号化されているため，オフロードされた IDS は VM のメモリ上にある OS データを解析して侵入を検知することができない．SEV からは IDS と攻撃者を区別することができないため，IDS にも暗号化された VM のメモリを復号する手段がないためである．第二に，オフロードされた IDS が VM を監視できたとしても，IDS 経由で機密情報が漏洩する恐れがある．IDS は VM から機密情報を含む可能性のあるデータを取得するため，クラウドの内部犯は IDS を攻撃することにより機密情報の一部を取得できるためである．

3. SEVmonitor

本稿では，SEV を用いてメモリが暗号化された VM に対して IDS オフロードを実現するシステム SEVmonitor を提案する．

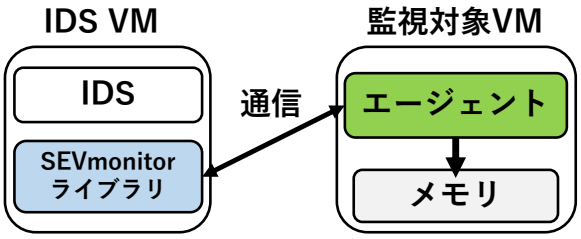
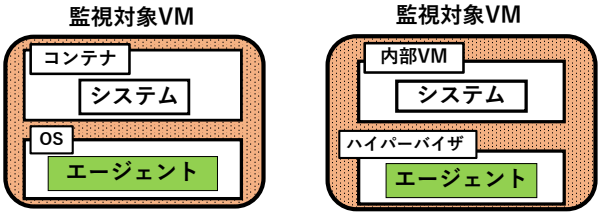


図 2: SEVmonitor のシステム構成
Fig. 2 The system architecture of SEVmonitor.



(a) コンテナによる隔離　(b) 内部 VM による隔離

図 3: 安全なエージェントの配置
Fig. 3 Secure deployment of agents.

3.1 システム構成

SEVmonitor のシステム構成を図 2 に示す．SEVmonitor は監視対象 VM の内部でメモリデータを取得するためのエージェントを安全に動作させる．エージェントとは VM 内にインストールされるソフトウェアのことである．エージェントを導入することにより，SEV を用いて VM のメモリが暗号化されていても，オフロードされた IDS はエージェント経由で VM のメモリデータを取得して OS データを解析することができる．

また，SEVmonitor は IDS も SEV を用いてメモリが暗号化された IDS VM 内で安全に実行する．これにより，攻撃者は IDS を攻撃することによって IDS が取得した監視対象 VM 内の機密情報を盗むことはできない．IDS VM に侵入されると IDS 内の機密情報が漏洩する恐れがあるが，IDS VM では IDS のみを動作させるため，監視対象 VM よりは侵入を防ぐのが容易である．

SEVmonitor は IDS VM 内の IDS に SEVmonitor ライブラリを提供する．このライブラリと監視対象 VM 内のエージェントとの間で仮想ネットワークまたは共有メモリを用いて通信を行う．IDS が監視対象 VM のメモリデータを取得するには，まず，SEVmonitor ライブラリ経由で取得したいメモリデータのアドレスをエージェントに送信する．エージェントは受信したアドレスに対応するメモリデータを取得し，SEVmonitor ライブラリに返送する．そして，IDS は SEVmonitor ライブラリが受信したメモリデータに対してアクセスを行う．仮想ネットワークや共有メモリはクラウドの内部犯によって盗聴される恐れがあるため，SEVmonitor は送信するアドレスやメモリデータを暗号化する．この暗号鍵は SEV でメモリが暗号化されたそれぞれの VM 内に格納されているため，内部犯は暗号化されたデータを復号することはできない．

3.2 エージェントの保護

監視対象 VM 内に配置されるエージェントは VM 内の監視対象システムに侵入されたとしても無効化されず，安全に動作し続けることができるようにしなければならない．SEVmonitor は監視対象 VM 内で監視対象システムを隔離環境に閉じ込め，その外側にエージェントを配置する．用いる隔離環境によって様々なトレードオフが存在するため，SEVmonitor は現在のところ，2 種類のエージェント配置を提供している．

1 つ目は，図 3(a) のように監視対象システムを監視対象 VM 内に作成したコンテナに閉じ込め，エージェントをその外側の OS カーネル内に配置する方法である．このエージェント配置では，OS レベルの軽量な仮想環境であるコンテナを用いるため，監視対象システムの性能がほとんど低下しない．また，OS の豊富な機能を用いることができるため，エージェントの開発が容易である．しかし，コンテナによる隔離は VM による隔離と比べて弱く，コンテナ経由で OS が攻撃を受けてエージェントが無効化される危険性がある．また，コンテナは OS に変更を加える権限を持っていないため，監視対象システムの機能が制限される．エージェントはコンテナだけを監視することもできるが，システム全体を監視することでコンテナの外で動作するコンテナエンジンなどへの攻撃も監視することができる．

2 つ目は，図 3(b) のように監視対象システムを監視対象 VM 内に作成した内部 VM 内に閉じ込め，エージェントをその外側のハイパーバイザ内に配置する方法である．VM はコンテナよりも隔離が強いため，エージェントをより安全に動作させることができる．また，監視対象システムが内部 VM 内の OS を自由に扱うことができるため自由度は高い．しかし，VM 内に内部 VM を作成するためのネストした仮想化のオーバヘッドが大きい．また，ハイパーバイザは最小限の機能しか持たないため，エージェントの開発は比較的難しい．

4. 実装

SEVmonitor を KVM 上で SEV を有効にした VM を用いて実装した．カーネル内エージェントは Linux 5.4 に実装し，ハイパーバイザ内エージェントは BitVisor および Xen 4.16 に実装した．

4.1 カーネル内エージェント

監視対象システムを VM 内のコンテナに隔離する場合，エージェントは VM 内の OS カーネルで動作する．そこで，エージェントをカーネルスレッドとして動作させる Linux カーネルモジュールを開発した．

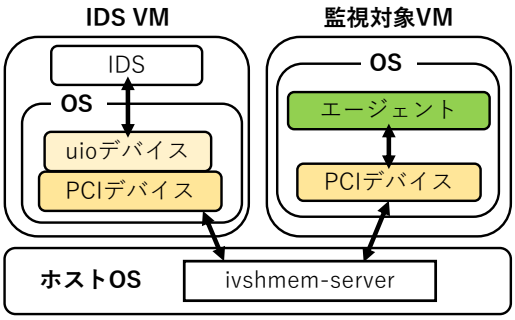


図 4: 共有メモリを用いた通信

Fig. 4 Communication between VMs using shared memory.

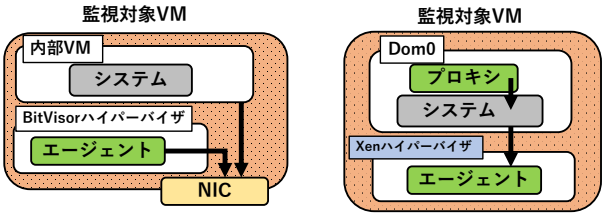
4.1.1 仮想ネットワークを用いた通信

エージェントは監視対象 VM と IDS VM 間の仮想ネットワークを用いて TCP/IP 通信を行う．エージェントの起動時に IDS からの接続要求を待ち受け，IDS から接続要求がきた時に接続を確立する．IDS からのメモリデータ取得要求を受信すると，エージェントはその要求に格納された仮想アドレスに対応する 4KB のメモリページのデータを取得し，そのデータを IDS に返送する．この際に，従来の IDS オフロードのようなアドレス変換を行う必要はなく，監視対象 VM 内のすべてのメモリに仮想アドレスを用いてアクセスすることができる．エージェントが送受信するデータは AES による暗号化を行う．

4.1.2 共有メモリを用いた通信

仮想ネットワークを用いた通信のオーバヘッドを減らすために，エージェントは VM 間の共有メモリを用いて高速な通信を行うことができる．SEVmonitor は ivshmem [4] を用いて図 4 のようにホスト OS 上のメモリ領域をそれぞれの VM にマップし，仮想的な PCI デバイスとして見せる．VM 内では ivshmem-uio ドライバ [5] を用いることによって，共有メモリにアクセスする．監視対象 VM では，共有メモリのために用いられる PCI デバイスのメモリをカーネルのメモリアドレス空間にリマップすることによって，エージェントが共有メモリに直接アクセスする．リマップの際にはカーネルのページテーブルエントリ（PTE）の C ビットが 0 に設定されるため，SEV によるメモリ暗号化の対象外となる．

一方，IDS VM ではユーザ空間で動作する IDS が共有メモリにアクセスできるようにするために，共有メモリのために用いられる PCI デバイスを uio デバイスとして提供する．uio はユーザ空間からデバイスのメモリにアクセスすることを可能にする Linux のインタフェースである．IDS が mmap システムコールを用いて uio デバイスのメモリをマップできるようにするために，ivshmem-uio ドライバ内で PCI デバイスのメモリを IDS プロセスのメモリアドレス空間にリマップする．このメモリが SEV によって暗号化されないようにするために，PTE の C ビットを 0 に設定する．



(a) BitVisor を用いた実装　(b) Xen を用いた実装

図 5: ハイパーバイザ内エージェント

Fig. 5 Agents in the hypervisor

4.2 ハイパーバイザ内エージェント

監視対象システムを VM 内の内部 VM に隔離する場合，エージェントは VM 内のハイパーバイザで動作する．ハイパーバイザとして BitVisor と Xen の 2 種類を用い，それぞれのハイパーバイザ内にエージェントを実装した．現在のところ，共有メモリを用いた通信には未対応である．

4.2.1 BitVisor を用いた実装

ネストした仮想化のオーバヘッドを削減するために，SEVmonitor は監視対象 VM の中で BitVisor [6] を動作させ，BitVisor 上で動作する内部 VM 内で監視対象システムを実行する．BitVisor は 1 台の VM のみを動かすように最適化されているため，KVM などの複数の VM をサポートした仮想化ソフトウェアより軽量に動作する．また，準パススルー型のアーキテクチャを採用しており，ハイパーバイザにおいてネットワークやストレージなどの処理が必要な場合だけ I/O を仮想化することができる．SEVmonitor では I/O を仮想化する必要がないため，監視対象システムの I/O 性能の低下を防ぐことができる．

エージェントは BitVisor が提供している軽量な TCP/IP スタックである lwIP を用いて IDS とのネットワーク通信を行う．そのために，図 5(a) のようにハイパーバイザと内部 VM の間で NIC を共有し，内部 VM とは異なる IP アドレスを割り当てる．BitVisor は lwIP の Raw API のみを提供しておりソケットを用いることができないため，コールバック関数を登録することによって通信処理を行う．IDS からメモリデータ取得要求を受信した時に登録したコールバック関数が呼び出される．エージェントは要求に格納された仮想アドレスに対応するメモリデータを内部 VM から取得し，そのデータを IDS に返送する．その際に軽量な暗号ライブラリである wolfSSL を用いて，仮想アドレスの復号化とメモリデータの暗号化を行う．

ハイパーバイザ内のエージェントは仮想アドレスのままでは内部 VM のメモリにアクセスできないため，アドレス変換を行う．まず，内部 VM のメモリ上にあるページテーブルからゲスト仮想アドレスに対応する PTE を探索し，その PTE に格納されているをゲスト物理アドレスを取得する．次に，ハイパーバイザ内にある内部 VM の EPT を用いてゲスト物理アドレスをホスト物理アドレスに変換

する．そして，変換したホスト物理アドレスを用いて内部VM のメモリデータを取得する．

現状では，KVM は SEV を有効にした VM 内でのハイパーバイザ実行に対応しておらず，BitVisor も起動しない．VM 内で BitVisor を動作させるためには，KVM が CPU の仮想化支援機構である SVM をエミュレートする必要がある．しかし，BitVisor が VM を管理するために用いる VMCB も SEV で暗号化されたメモリ上にあるため，KVM がアクセスすることはできない．VMCB が暗号化されないように PTE の C ビットを設定することで，SEV で暗号化された VM 内でも KVM を動作させられることが報告されている [7]．そのため，BitVisor を動作させることも可能であると考えられるが，現在のところ未実装である．

4.2.2 Xen を用いた実装

ネストした仮想化のオーバヘッドを削減する別の手法として，SEVmonitor は監視対象 VM の中で Xen [8] を動作させ，Xen の準仮想化 VM の中で監視対象システムを実行する．準仮想化 VM は OS を改変することによって VM 内での効率の良い OS 実行を可能にする．CPU の仮想化支援機構を用いないため，KVM が VM 内の Xen に対して SVM をエミュレートする必要がなく，BitVisor よりも SEV を有効にした VM 内で動作させるのが容易である．Xen を動作させるにはハイパーバイザの SEV 対応が必要となるが，現在のところ未実装である．

SEVmonitor は内部 VM として Xen の Dom0 を用いて監視対象システムを動作させる．Dom0 は Xen の起動時に常に起動される準仮想化 VM であり，他の VM を管理するために用いられる．Dom0 は CPU とメモリのみを仮想化しており，I/O の仮想化は行わないため，監視対象システムの I/O 性能の低下を防ぐことができる．Xen は DomU と呼ばれる通常の VM を準仮想化を用いて作成することもできるが，DomU の I/O は仮想化されるため性能への影響が大きい．Dom0 はシステム全体を管理する権限を持っているが，ハイパーバイザは Dom0 から保護されている．また，Dom0 はハイパーバイザを置き換えて起動する権限も持っているが，リモートアテステーションを用いてハイパーバイザの正常起動を確認することでハイパーバイザの置き換えを検知することができる．

Xen を用いる場合，図 5(b) のように IDS とハイパーバイザ内のエージェントは Dom0 内の OS 上で動作するプロキシを経由して通信を行う．これは BitVisor とは異なり，Xen ハイパーバイザが通信機能を持っていないためである．プロキシが IDS からのメモリデータ取得要求を受信すると，ハイパーバイザに対してハイパーコールを呼び出すことによって Dom0 のメモリデータを取得し，そのデータを IDS に返送する．プロキシが扱うデータは IDS とハイパーバイザ間で暗号化されるため，Dom0 への侵入者に監視内容が漏洩することはない．一方，プロキシ経由でネットワーク通信を行うと監視性能が低下する可能性がある．このオーバヘッドはハイパーバイザと IDS の間で共有メモリを用いることで削減することができる．

プロキシから呼び出されるハイパーコールは仮想アドレスを指定して Dom0 のメモリデータを安全に取得する．まず，Dom0 のメモリ上にあるページテーブルを用いて仮想アドレスから疑似物理フレーム番号への変換を行う．次に，ハイパーバイザ内の P2M テーブルを用いて疑似物理フレーム番号からマシンフレーム番号に変換する．そのフレーム番号に対応するページをマップすることにより，Dom0 のメモリにアクセスする．ハイパーコールは暗号化された仮想アドレスを受け取り，取得したメモリデータを暗号化して返すが，この暗号化については現在実装中である．

4.3 SEVmonitor ライブラリ

IDS が OS データを解析してシステムの監視を行うのを容易にするために，SEVmonitor は LLView フレームワーク [9] を用いる．LLView は Linux のソースコードを用いた IDS プログラムの開発を可能にする．開発した IDS プログラムは LLVM を用いてコンパイルされ，出力された中間表現の中のメモリロード命令の変換が行われる．ロード命令の直前に SEVmonitor ライブラリを呼び出すコードを挿入してロード命令の対象アドレスを渡し，SEVmonitor ライブラリが IDS に対して透過的に監視対象 VM 内のエージェントとの通信を行う．そして，エージェント経由で取得したメモリデータを IDS のロード命令で読み込む．

SEVmonitor はネットワーク通信用と共有メモリ用の 2 種類の SEVmonitor ライブラリを提供する．ネットワーク通信を用いる場合，SEVmonitor ライブラリは初期化時にエージェントとの TCP 接続を確立する．LLView が挿入したコードによって呼び出されると，ロード命令の対象アドレスを AES で暗号化したのちにエージェントにメモリデータ取得要求を送信する．暗号化されたメモリデータを受信すると AES で復号する．共有メモリを用いる場合，SEVmonitor ライブラリはロード命令の対象アドレスを AES で暗号化して共有メモリに書き込み，共有メモリ内の書き込みフラグをセットすることによってエージェントへの通知を行う．その後，エージェントからの書き込みフラグがセットされるのをビジーウェイトで待ち，セットされたら暗号化されたメモリデータを読み込んで復号する．

5. 実験

SEV でメモリが暗号化された VM から SEVmonitor を用いて OS データが取得できるかどうかを確認し，取得にかかる時間を調べた．エージェントを監視対象 VM の OS カーネル内に配置した場合には仮想ネットワークと共有メモリについて実験を行い，BitVisor および Xen のハイ

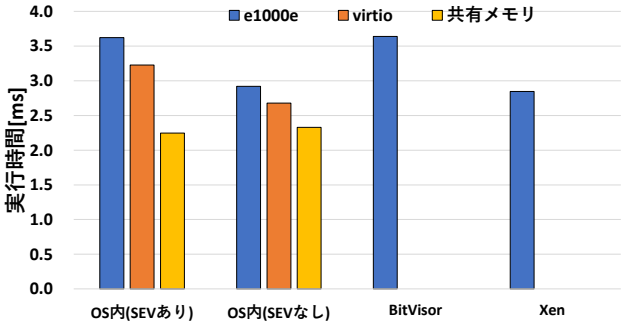


図 6: OS バージョンの取得性能

Fig. 6 The performance of obtaining the OS version.

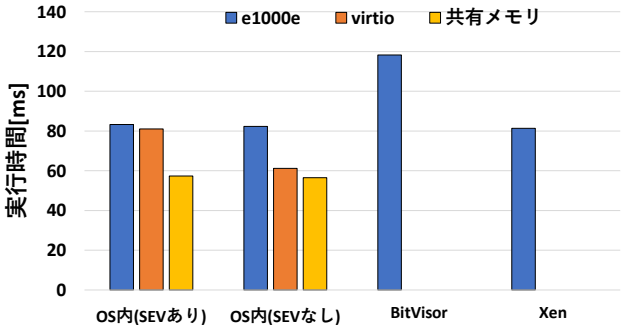


図 7: プロセス一覧の取得性能

Fig. 7 The performance of obtaining the process list.

パーバイザ内に配置した場合には仮想ネットワークについてのみ実験を行った．また，SEV を有効にした場合と無効にした場合での実験を行い，SEV による性能への影響も調査した．実験に用いたマシンの CPU は AMD EPYC 7262，メモリは 128GB，ネットワークは 10GbE であった．ホスト OS として Linux 5.11.0，仮想化ソフトウェアとして QEMU-KVM 6.2 を動作させた．監視対象 VM と IDS VM には，それぞれ仮想 CPU を 2 個，メモリを 2GB 割り当て，ゲスト OS として Linux 5.4.0 を動作させた．

5.1 動作確認

まず，監視対象システム内で動作している OS のバージョン情報を取得する IDS を実行した．この IDS は Linux カーネルの linux_banner 変数に格納されている文字列を取得する．そのために，エージェントに対して要求を 1 回送信し，4KB のメモリデータを取得した．実験の結果，エージェントを監視対象 VM 内の OS カーネルとハイパーバイザのどちらに配置した場合でも，OS バージョンが取得できることを確認した．

次に，監視対象システム内で動作しているプロセス一覧の情報を取得する IDS を実行した．この IDS は Linux カーネルの init_task 変数からプロセスリストをたどり，プロセスの ID と名前を取得する．コンテナおよび BitVisor の内部 VM に隔離した監視対象システムでは 119 個のプロセスが動作していたため，エージェントに対して要求を 119 回送信し，合計で 476KB のメモリデータを取得した．一方，Xen の内部 VM に隔離した監視対象システムでは 127 個のプロセスが動作していたため，要求を 127 回送信した．実験の結果，エージェントを監視対象 VM 内の OS カーネルとハイパーバイザのどちらに配置した場合でも，プロセス一覧が取得できることを確認した．ただし，Xen を用いた場合は通信データの暗号化は行っていない．

5.2 カーネル内エージェントの性能

コンテナで隔離した監視対象システム内の OS データを OS カーネル内のエージェントから取得するのにかかる時間を測定した．この実験では，監視対象 VM の仮想 NIC として e1000e および virtio を用いてネットワーク通信を行った場合と，共有メモリを用いて通信を行った場合とを比較した．また，それぞれ SEV を有効にした場合としなかった場合の取得時間も測定した．OS バージョンの取得時間を図 6 に，プロセス一覧の取得時間を図 7 に示す．

SEV を有効にした場合の OS バージョンの取得性能については，準仮想化 I/O である virtio を用いたネットワーク通信と比較して，共有メモリを用いた場合には $1ms$ 高速になることが分かった．一方，e1000e のデバイスエミュレーションを行った場合と比べると，virtio を用いることによる高速化は $0.4ms$ であった．SEV を有効にすることにより，仮想ネットワークを用いた場合には $0.7ms$ 遅くなったが，共有メモリを用いた場合には影響を受けなかった．

一方，プロセス一覧の取得性能については，SEV を有効にした場合，仮想ネットワークを用いた場合に比べて共有メモリを用いた場合は 25%高速になり，e1000e と virtio ではほとんど性能差がないことが分かった．一方，SEV を無効にしても virtio を用いた場合以外は性能が変わらなかった．virtio を用いた場合は SEV を有効にすることで性能が 25%低下した．この原因として，ホスト OS が virtio のバッファにアクセスできるようにするために，暗号化されないメモリ領域へのデータコピーが行われている可能性がある．

5.3 ハイパーバイザ内エージェントの性能

BitVisor および Xen の内部 VM で隔離した監視対象システム内の OS データをハイパーバイザ内のエージェントから取得するのにかかる時間を測定した．この実験では，監視対象 VM の仮想 NIC として e1000e を用いてネットワーク通信を行った．OS バージョンの取得時間は図 6 に，プロセス一覧の取得時間は図 7 に示されている．

OS バージョンの取得性能については，BitVisor を用いた場合は SEV を有効にしてネットワーク通信を用いるカーネル内エージェントと同程度であった．SEV を無効にした場合と比較すると，カーネル内エージェントより $0.7ms$ 遅くなった．一方，Xen を用いた場合は SEV を無効にして

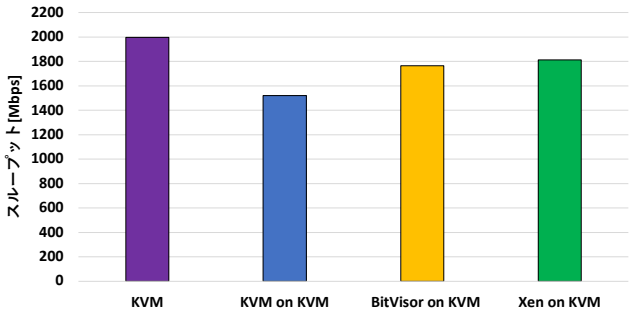


図 8: 内部 VM のネットワーク性能

Fig. 8 The network performance of internal VMs.

ネットワーク通信を用いるカーネル内エージェントと同程度の性能であった．ただし，通信データの暗号化をまだ行えていないため，暗号化によって多少性能が低下する可能性がある．これらの結果より，BitVisor を用いるより Xen を用いる方が OS データの取得性能は高いことが分かった．Xen を用いた実装では Dom0 でプロキシを動作させているため，ハイパーコールを利用しないようにすればさらなる高速化が期待できる．

プロセス一覧の取得性能については，BitVisor を用いた場合はネットワーク通信を用いるカーネル内エージェントと比較して 50%遅くなった．一方，Xen を用いた場合はネットワーク通信を用いるカーネル内エージェントと同程度の性能であった．これらの結果より，BitVisor を用いるよりも Xen を用いる方がプロセス一覧の取得性能が 30%高いことが分かった．

5.4 内部 VM の性能

KVM 上で BitVisor と Xen を用いることによりネストした仮想化のオーバヘッドが削減できるかどうかを調べるために，内部 VM のネットワーク性能を測定した．この実験では外部ホストから iperf を用いて TCP/IP のスループットを測定した．比較として，KVM 上で KVM を動作させた場合と通常の VM を動作させた場合についても測定した．実験結果を図 8 に示す．KVM 上で BitVisor と Xen を用いた場合には KVM のみを用いた場合と比較して 200$Mbps$ スループットが低下したが，KVM 上で KVM を用いた場合よりも 300$Mbps$ 高速であることが分かった．また，BitVisor と Xen のスループットは同程度であることも分かった．

6. 関連研究

Intel SGX を用いて作成されたエンクレイヴ内部でエージェントを動作させる研究がいくつか行われている．SGX はエンクレイヴと呼ばれる保護領域をプロセス内に作成してメモリを保護することができる．SGX を用いた VM マイグレーション [10] や MigSGX [11] では，VM やコンテナのマイグレーション時にエンクレイヴ内部で動作するエージェントがエンクレイヴの状態を外部に保存する．ただし，エージェントは保護されていないため，エンクレイヴ内で信頼できないサービスが動作している場合には安全に実行することはできない．

Ryoan [12] は Google NaCl を用いてエンクレイヴ内にサンドボックスを構築し，コード検査とランタイムチェックを行うことでサービスを安全に実行する．そのため，サンドボックス外部にエージェントを配置することにより，エンクレイヴ内でエージェントを安全に実行することができる．同様に，AccTEE [13] はエンクレイヴ内で WebAssembly を用いてサンドボックスを構築し，サービスを安全に実行する．そして，サンドボックス外部にエージェントを配置して，サービスが使用したリソースをエンクレイヴ内に安全に記録する．しかし，SEV を用いて保護された VM 内の監視対象システム全体に対して NaCl や WebAssembly を適用するのは難しい．

SGX を用いて IDS を保護する手法も提案されている．S-NFV [14] はネットワークベース IDS である Snort の内部状態とそれを扱うコードをエンクレイヴ内に配置する．エンクレイヴ外部に対して安全な API を提供し，エンクレイヴ内のコードを呼び出すことによって内部状態を利用する．SEC-IDS [15] は Graphene-SGX ライブラリ OS [16] を用いて Snort 全体をエンクレイヴ内で動作させる．エンクレイヴ内でネットワークパケットを取得するために DPDK を用いる．これらのシステムでは，パケット取得中に攻撃者に書き換えられることは想定していない．

SGmonitor [17] はホストベース IDS をエンクレイヴ内で実行する．IDS はハイパーバイザ経由で VM のメモリデータを取得し，OS データを解析して監視を行う．SCwatcher [18] は SCONE [19] や Occlum [20] を用いて OS 標準インタフェースを提供することで，エンクレイヴ内で既存のホストベース IDS を実行可能にする．また，VM 内のシステム情報を返す proc ファイルシステムをエンクレイヴ内で提供する．これらのシステムでは VM のメモリデータを安全に取得するために，クラウドの管理下にあるハイパーバイザを信頼する必要がある．

x86 の動作モードの一つであるシステムマネジメントモード（SMM）を用いて IDS を安全に実行する手法も提案されている．HyperGuard [21] は SMM を用いてハイパーバイザの整合性を安全にチェックする．HyperSentry [22] は SMM を用いてハイパーバイザにエージェントを挿入し，エージェントがシステムの監視を行う．HyperCheck [23] は SMM を用いてリモートホストにメモリデータを送信し監視を行う．しかし，SMM によるプログラムの実行は低速であり，SMM を用いた監視を行う際にはシステム全体を停止させる必要がある．また，SMM で動作するコードは BIOS 内に実装する必要がある．

一方，RemoteTrans [24] はクラウドの外部に IDS をオフ

ロードし，クラウド内で動作する VM を監視する．SEVmonitor と同様に，IDS はクラウドのハイパーバイザ内のエージェントと通信して，VM の中の指定したメモリデータを取得することで監視を行う．しかし，クラウド内でエージェントを安全に実行するためにクラウドの管理下にあるハイパーバイザを信頼する必要がある．

7. まとめ

本稿では，SEV を用いてメモリが暗号化された VM に対して安全な IDS オフロードを実現するシステム SEVmonitor を提案した．SEVmonitor は監視対象 VM 内で監視対象システムをコンテナや内部 VM などの隔離環境に閉じ込め，その外側でエージェントを安全に動作させる．SEV で保護された別の VM 内で動作する IDS は，仮想ネットワークまたは共有メモリを用いてエージェントと暗号通信を行うことで VM のメモリデータを取得する．実験により，メモリが暗号化された VM から OS データを取得できることを確認し，その取得性能を調べた．

今後の課題は，IDS とハイパーバイザ内エージェントが共有メモリを用いた通信を行えるようにし，監視性能の向上を図ることである．また，SEV で保護された VM 内で BitVisor と Xen を動作させられるようにすることも必要である．

謝辞 本研究の一部は，JST，CREST，JPMJCR21M4 の支援を受けたものである．

参考文献

[1] Advanced Micro Devices, I.: Secure Encrypted Virtualization API Version 0.24 (2020).
[2] Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Instrusion Detection, *In Proc. Network and Distributed Systems Security Symp.*, pp. 191–206 (2003).
[3] IPA: 情報セキュリティ 10 大脅威,（オンライン），入手先 ⟨https://www.ipa.go.jp/security/vuln/10threats2022.html⟩.
[4] The QEMU project developers: Inter-VM Shared Memory device, (online), available from ⟨https://qemu-project.gitlab.io/qemu/system/ivshmem.html⟩.
[5] Macdonell, C.: ivshmem-uio, (online), available from ⟨https://github.com/shawnanastasio/ivshmem-uio⟩.
[6] Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E., Kono, K., Chiba, S., Shinjo, Y. and Kato, K.: BitVisor: A Thin Hypervisor for Enforcing I/O Device Security, *In Proc. Int. Conf. Virtual Execution Environment*, pp. 121–135 (2009).
[7] Christopherson, S.: KVM: SVM: Treat SVM as unsupported when running as an SEV guest, `https://www.spinics.net/lists/kvm/msg234582.html` (2021).
[8] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the Art of Virtualization, *In Proc. Symp. Operating Systems Principles*, pp. 164–177 (2003).
[9] Ozaki, Y., Kanamoto, S., Yamamoto, H. and Kourai, K.: Detecting System Failures with GPUs and LLVM, *In Proc. Asia Pacific Workshop on Systems* (2019).
[10] Gu, J., Hua, Z., Xia, Y., Chen, H., Zang, B., Guan, H. and Jinming, L.: Secure Live Migration of SGX Enclaves on Untrusted Cloud, *In Proc. Int. Conf. Dependable Systems and Networks* (2017).
[11] Nakashima, K. and Kourai, K.: MigSGX: A Migration Mechanism for Containers Including SGX Applications, *In Proc. Int. Conf. Utility and Cloud Computing* (2021).
[12] Tyler, H., Zhiting, Z., X. Yuanzhong, P. S. and Emmett, W.: Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data, *In Proc. USENIX Symp. Operating Systems Design and Implementation* (2016).
[13] Goltzsche, D., Nieke, M., Knauth, T. and Kapitza, R.: AccTEE: A WebAssembly-based Two-way Sandbox for Trusted Resource Accounting, *In Proc. Int. Middleware Conf.* (2019).
[14] Shih, M., Kumar, M., Kim, T. and Gavrilovska, A.: Securing NFV States by Using SGX, *In Proc. Int. Workshop on Security in Software Defined Networks and Network Function Virtualization* (2016).
[15] Kuvaiskii, D., Chakrabarti, S. and Vij, M.: Intrusion Detection System with Intel Software Guard Extension, *arXiv:1802.00508* (2018).
[16] Tsai, C., Porter, D. and Vij, M.: Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX, *In Proc. USENIX Annual Technical Conf.* (2017).
[17] Nakano, T. and Kourai, K.: Secure Offloading of Intrusion Detection Systems from VMs with Intel SGX, *In Proc. Int. Conf. Cloud Computing*, pp. 297–303 (2021).
[18] Kawamura, T. and Kourai, K.: Secure Offloading of User-level IDS with VM-compatible OS Emulation Layers for Intel SGX, *In Proc. Int. Conf. Cloud Computing*, pp. 157–166 (2022).
[19] Arnautov, S., Trach, B., Gregor, F., Knauth, T., Martin, A., Priebe, C., Lind, J., Muthukumaran, D., O'Keeffe, D., Stillwell, M., Goltzsche, D., Eyers, D., Kapitza, R., Pietzuch, P. and Fetzer, C.: SCONE: Secure Linux Containers with Intel SGX, *In Proc. Symp. Operating System Design and Implementation*, pp. 689–705 (2016).
[20] Shen, Y., Tian, H., Chen, Y., Chen, K., Wang, R., Xu, Y., Xia, Y. and Yan, S.: Occlum: Secure and Efficient Multitasking Inside a Single Enclave of Intel SGX, *In Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems*, pp. 955–970 (2020).
[21] Rutkowska, J. and Wojtczuk, R.: Preventing and detecting Xen hypervisor subversions, *Blackhat Briefings USA* (2008).
[22] Azab, A., Ning, P., Wang, Z., Jiang, X., Zhang, X. and Skalsky, N.: HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity, *In Proc. Conf. Computer and Communications Security*, pp. 38–49 (2010).
[23] Wang, J., Stavrou, A. and Ghosh, A.: HyperCheck: A hardware-assisted Integrity Monitor, *In Proc. Int. Symp. Recent Advances in Intrusion Detection*, pp. 157–177 (2010).
[24] Kourai, K. and Juda, K.: Secure Offloading of Legacy IDSes Using Remote VM Introspection in Semi-trusted Clouds, *In Proc. Int. Conf. Cloud Computing*, pp. 43–50 (2016).