Vol.2021-OS-153 No.2
2021/7/20

AMD SEVを用いてメモリが暗号化されたVMに対するIDSオフロード

能野 智玄[1]　光来 健一[1]

概要：IaaS 型クラウドの普及により，クラウドの仮想マシン（VM）においても重要なデータが扱われるようになっている．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あるが，AMD SEV を用いて VM のメモリを透過的に暗号化することにより，メモリの盗聴による情報漏洩を防ぐことができる．一方，VM 内に侵入されると SEV による VM の保護は機能しなくなるため，侵入検知システム（IDS）を用いて攻撃を検知する必要がある．しかし，VM の外に IDS をオフロードして安全に実行しようとしても，暗号化された VM のメモリ上のデータを監視することはできない．本稿では，SEV を用いてメモリが暗号化された VM 内でエージェントを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 は監視対象 VM の内部でエージェントを安全に動作させ，IDS がエージェントからメモリデータを取得することによって VM の監視を行う．IDS も SEV によって保護された別の VM 内で動作させることで，IDS 経由での情報漏洩も防ぐ．SEVmonitor を KVM と Linux を用いて実装し，監視対象 VM の OS データを取得する性能を調べた．

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，クラウド内の内部犯から 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きた．この問題に対して，AMD 製の CPU では VM のメモリを保護するために，Secure Encrypted Virtualization (SEV) と呼ばれる透過的なメモリ暗号化が提供されている．SEV は VM のメモリを暗号化し，VM 内でアクセスする時にだけ復号する．SEV を用いることによって，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，侵入検知システム（IDS）を用いて VM 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 [1] という手法が用いられている．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い．

本稿では，SEV を用いてメモリが暗号化された VM に対する 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．そして，オフロードされた IDS がエージェントから 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．エージェントの配置には様々なトレードオフがあるため，SEVmonitor はカーネル内エージェント，ネストした仮想化を用いたエージェント，OS から隠されたエージェントの 3 つを提供する．

我々は SEVmonitor を KVM と Linux を用いて実装した．カーネル内のエージェントはカーネルモジュールとして実装し，VM 間の仮想ネットワークまたは共有メモリを用いて IDS との通信を行う．通信データは SEVmonitor が独自の暗号化を行うことによって，エージェントと IDS 間でやりとりできるようにしつつ，クラウドの内部犯への情報漏洩を防ぐ．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の一覧の取得ができることを確認した．TCP 通信を用いると VM からのプロセス情報の取得に $82ms$ かかり，共有メモリを用いると $48ms$ かかることが分かった．

以下，2 章では SEV を用いて暗号化された VM の監視について述べる．3 章では VM の内部でエージェントを安

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

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 章で関連する研究について述べ，7 章で本稿をまとめる．

2. SEV を用いて保護された 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 [2]．内部犯から 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可能性がある．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 CPU の中のメモリコントローラに組み込まれた暗号化エンジンが VM ごとに異なる暗号鍵・復号鍵を用いてメモリの暗号化・復号化を行う．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．

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 は機能しない．なぜなら，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．メモリから直接，機密情報を盗むことができなくても，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 へ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 [1]．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

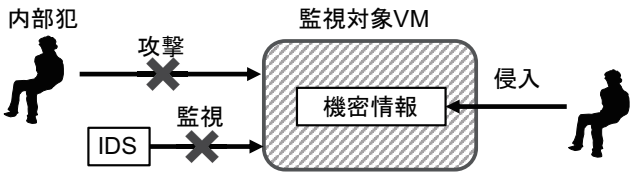


図 1: SEV を用いてメモリが暗号化された VM への攻撃

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 に対して安全な IDS オフロードを実現するシステム SEVmonitor を提案する．

3.1 システムの概要

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，VM のメモリが暗号化されていてもオフロードされた IDS が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 専用 VM 内で安全に実行する．これにより，IDS を攻撃することによって IDS が取得した機密情報を盗むことはできない．IDS VM に侵入されると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．このようにして，クラウドの内部犯と VM 内への侵入者の双方からクラウド内の VM を守ることができる．

SEVmonitor は IDS VM 内の IDS に提供されるライブラリと監視対象 VM 内のエージェントとの間で，仮想ネットワーク通信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 ライブラリ経由でその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，ライブラリに返送する．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，送信するアドレスやメモリデータは暗号化する．この暗号鍵はメモリが暗号化された VM 内に格納されているため，攻撃者が暗号化されたデータを復号することはできない．

3.2 エージェントの配置

監視対象 VM 内に配置されるエージェントは，攻撃者に VM 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．そのため，

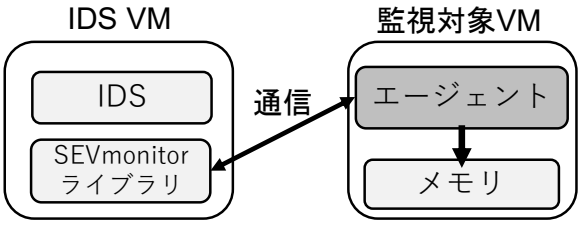


図 2: SEVmonitor のシステム構成

監視対象 VM 内のどこに配置するかが重要になる．エージェントの配置には様々なトレードオフが存在するため，SEVmonitor は現在のところ，3つの配置を提供している．以下では，それぞれの配置が想定する脅威モデルと利点および欠点について述べる．

第一に，図 3(a) のように OS カーネル内にエージェントを配置することができる．この配置では，OS カーネルが攻撃されるとエージェントが無効化されてしまう恐れがあるため，OS カーネルに脆弱性がないことを仮定している．エージェントがカーネルモジュールなどの形で OS カーネルの一部となるため，OS の豊富な機能を使うことができるという利点がある．一方，監視対象システムに組み込む必要があるため，システムの管理コストが上昇する可能性がある．

第二に，図 3(b) のようにネストした仮想化 [3] を用いた監視対象システムを内部 VM の中に閉じ込め，その外側にエージェントを配置することができる．エージェントと内部 VM が同一の鍵で暗号化された監視対象 VM 内で動作するため，エージェントが監視対象システムのメモリにアクセスすることができる．この配置では，内部 VM の下で動作するハイパーバイザは攻撃を受けないことを仮定している．攻撃者が監視対象システム内へ侵入したとしてもエージェントは無効化されず，攻撃を検知できるという利点がある．しかし，ネストした仮想化によるオーバヘッドにより，監視対象システムの性能が低下する．

第三に，図 3(c) のように監視対象システムの管理外にエージェントを配置することができる．OS 起動前にエージェントを配置することにより，OS がエージェントを認識できないようにする．この配置では，攻撃者がエージェントを見つけられないことを仮定している．OS カーネルが攻撃されてもエージェントが無効化されにくく，監視対象システムに対するオーバヘッドが小さいという利点がある．しかし，OS の機能を用いずにエージェントを実装するのが難しい．

4. 実装

我々は SEVmonitor を KVM と Linux を用いて実装した．3 章の 3 つのエージェント配置のうち，OS カーネル内で動作するエージェントを Linux 5.4 に実装した．ネストした仮想化を用いるエージェントも BitVisor[4] を用いて実装中である．また，エージェントと通信して VM のメモリデータを取得する SEVmonitor ライブラリも実装した．

4.1 SEV を有効にした VM

SEV を有効にした VM を作成できるようにするために，ホスト Linux の起動オプションで SEV およびメモリ暗号化を有効にする．SEV は UEFI BIOS でのみ動作するため，VM のマシンタイプには Q35 を指定し，BIOS として OVMF を用いる．SEV は暗号化されたメモリページがホスト OS によってスワップアウトされないことを仮定しているため，VM のメモリページがホストのメモリにロックされるように設定する．libvirt を用いて SEV を有効にするために Launch Security のタイプを sev に設定する．SEV による暗号化はページ単位で制御されるため，暗号化するかどうかを表すページテーブルエントリの C ビットの位置を指定する．

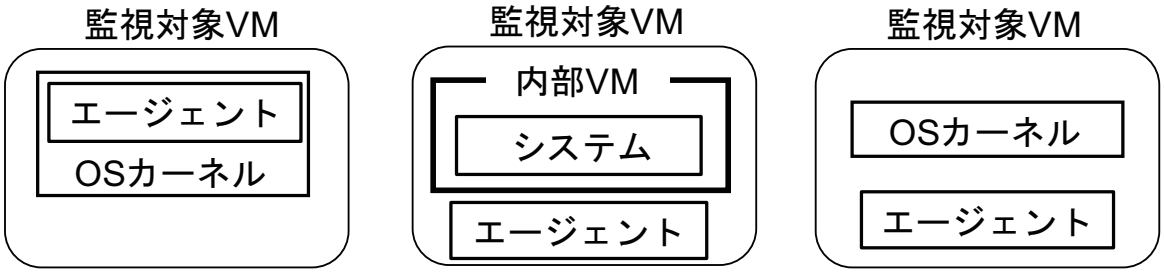
4.2 カーネル内エージェント

カーネルスレッドとしてエージェントを動作させる Linux カーネルモジュールを作成した．このエージェントは仮想ネットワークまたは共有メモリを用いて SEVmonitor ライブラリとの通信を行う．

4.2.1 仮想ネットワークを用いるエージェント

このエージェントは TCP を用いたネットワーク通信を行う．IDS を起動した時に SEVmonitor ライブラリが監視対象 VM の IP アドレスとポート番号を指定してエージェントとの接続を確立する．IDS が監視対象 VM のメモリデータを必要とした時には，SEVmonitor ライブラリがその仮想アドレスをエージェントに送信する．エージェントはその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，SEVmonitor ライブラリに返送する．エージェントは監視対象 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できるため，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い．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．AES は 16 バイト単位で暗号化を行うため，SEVmonitor ライブラリが 8 バイトの仮想アドレスを送信する際には 8 バイトの余分なデータを付与する．

エージェントは接続要求とメッセージ受信を待つのにブロッキングモードまたはノンブロッキングモードを選択することができる．ブロッキングモードでは，接続要求を受け付ける kernel_accept 関数とメッセージを受け取る kernel_recvmsg 関数の中で待機する．常に通信要求を待っているため，通信が行われると即座に応答することができる．このモードではエージェントが通信以外の要求を処理することはできないが，ソケットにタイムアウトを設定することによって一定時間ごとに要求を受け付けられるよう



(a) カーネル内　(b) ネストした仮想化　(c) OS からの隠蔽

図 3: エージェントの配置

にすることができる．

ノンブロッキングモードでは，接続要求が来るまで kernel_accept 関数を繰り返し実行し，接続を確立した後は kernel_recvmsg 関数を繰り返し実行する．その合間にエージェントが他の処理を行うことができるため，ミリ秒単位でのタイムアウト時にしか他の処理を行えないブロッキングモードよりも応答性を高めることができる．しかし，カーネル内でビジーループを行うと CPU を占有してしまうため，マイクロ秒単位のスリープを入れることで CPU を占有しないようにする．

4.2.2 共有メモリを用いるエージェント

このエージェントはネットワーク通信のオーバヘッドを減らすために VM 間の高速な共有メモリを用いる．QEMU の ivshmem[5] という機能を用いて，図 4 のようにホスト OS 上のメモリ領域を複数の VM にマップし，仮想的な PCI デバイスとして見せる．ホスト OS 上で ivshmem-server を実行することによって，VM 間で共有メモリの同期をとることができる．また，共有メモリへの書き込み時に他の VM に割り込みを送ることもできる．

VM 内では ivshmem-uio ドライバ [6] 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ことができる．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．そのため，他の VM とメモリを共有することができる．

一方，IDS VM ではユーザ空間の SEVmonitor ライブラリ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．既存の ivshmem-uio ドライバが提供する uio デバイスは read や write システムコールを用いる読み書きしかサポートしていなかった．

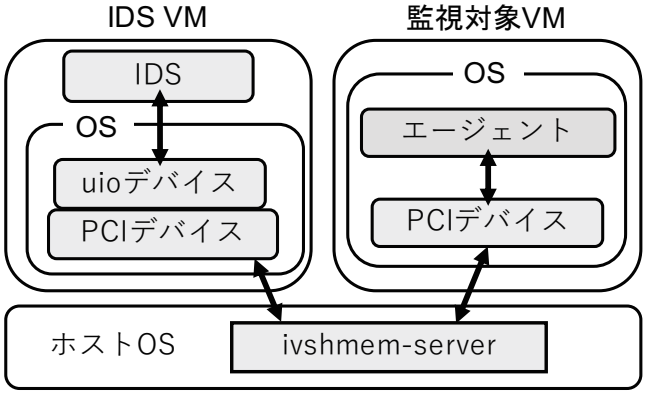


図 4: 共有メモリを用いた通信

そこで，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することにより，IDS が共有メモリに直接アクセスできるようにした．そのために，ドライバ内で PCI デバイスのメモリをプロセスのメモリアドレス空間にリマップする．しかし，Linux カーネルはプロセスのアドレス空間へのリマップ時には，プロセスのページテーブルエントリの C ビットを 0 にしない．共有メモリが暗号化されないようにするために，この場合にも C ビットを 0 に設定する．この機能については現在実装中であるため，共有メモリを使う場合には IDS VM のメモリ暗号化は行っていない．

現在のところ，ivshmem の割り込みがうまく使えていないため，共有メモリ上のフラグを監視することによりメッセージの送受信の同期をとっている．フラグの監視はスリープしながらループすることによって行う．共有メモリのフォーマットを図 5 に示す．最初の 1 バイトには SEVmonitor ライブラリ用の書き込みフラグ，次の 1 バイトにはエージェント用の書き込みフラグを格納し，8 バイト整列にするために次の 6 バイトを空ける．次の 8 バイトには仮想アドレスを格納し，AES を用いて 16 バイト単位で暗号化するために次の 8 バイトを空ける．そして，次の 4096 バイトにメモリデータを格納する．共有メモリは他の VM やホスト OS から覗き見られる可能性があるため，AES を用いて，仮想アドレスとメモリデータを暗号化す

フラグ 1	フラグ 2	未使用	仮想アドレス	未使用	メモリデータ

図 5: 共有メモリのフォーマット

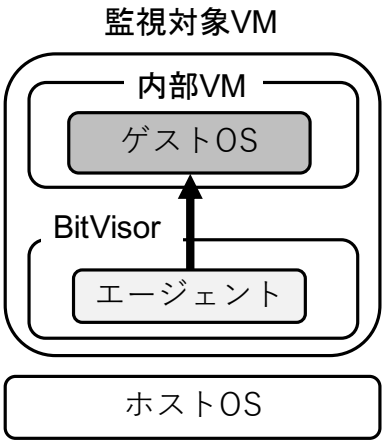


図 6: BitVisor を用いたエージェントの配置

る．フラグから情報が漏洩することはないため，フラグは暗号化しない．

4.3 ネストした仮想化を用いたエージェント

ネストした仮想化を用いて監視対象 VM の中に内部 VM を作るために，VM の中で BitVisor を動作させ，BitVisor によって作成される VM の中で監視対象システムを実行する．BitVisor を用いた監視対象 VM のシステム構成を図 6 に示す．BitVisor は準パススルー型のアーキテクチャを用いており，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．また，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，VM 内でも他の仮想化ソフトウェアより軽量に動作すると考えられる．現状では，SEV を有効にした VM では BitVisor が起動しない．原因は特定できていないが，BitVisor 内のいくつかのページについて SEV による暗号化の除外が必要である可能性がある．

エージェントは BitVisor のハイパーバイザ内に実装し，軽量な TCP/IP スタックである lwIP を用いて SEVmonitor ライブラリとのネットワーク通信を行う．そのために，内部 VM との間で NIC を共有し，内部 VM とは異なる IP アドレスを割り当てる．また，共有メモリ用の PCI デバイスのメモリをリマップすることで共有メモリを用いた通信を実現する．内部 VM のメモリデータには，仮想アドレスをゲスト物理アドレスに変換し，さらにホスト物理アドレスに変換することでアクセスする．このエージェントは現在実装中である．

4.4 IDS による OS データの監視

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 [7] を用いる．LLView は Linux のソースコードを用いて IDS プログラムを開発することを可能にする．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ロード命令の変換が行われる，ロード命令が VM のメモリからデータを読み込むようにすることによって，IDS が VM 内のシステムを透過的に監視できるようにする．SEVmonitor では VM のメモリデータを取得する処理をエージェントとの通信に置き換えた．

5. 実験

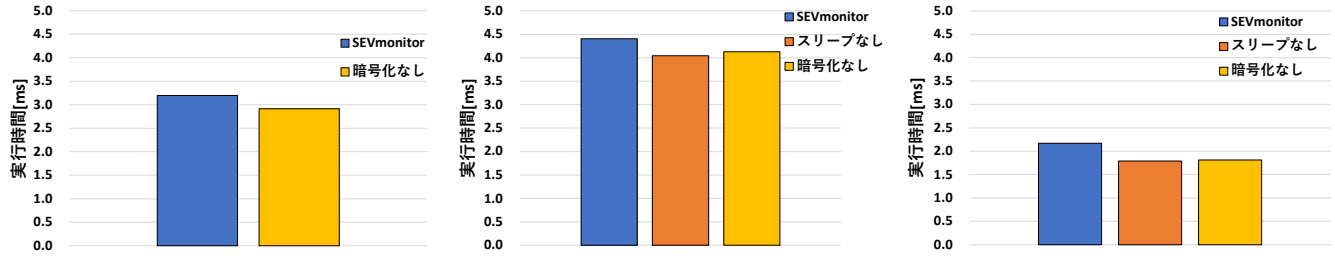
SEVmonitor を用いて，メモリが暗号化された VM から OS データが取得できることの確認を行い，取得にかかる時間を調べた．取得性能については，IDS とエージェント間の TCP 通信にブロッキングモードを用いた場合とノンブロッキングモードを用いた場合および，共有メモリを用いた場合について測定した．ノンブロッキングモードまたは共有メモリを用いる場合については，データ受信処理のループ中に 200 μs のスリープを入れた．比較として，スリープなしでビジーループを行う場合と，データの暗号化を行わない場合についても測定した．実験に用いたマシンの CPU は AMD EPYC 7262，メモリは 128GB であった．ホスト OS として Linux 5.4.0，仮想化ソフトウェアとして QEMU-KVM 4.2 を動作させた．IDS VM と監視対象 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 OS のバージョン情報の取得

監視対象 VM から 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して表示する．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．その結果，OS のバージョン情報が表示されることを確認できた．

バージョン情報の取得にかかった時間を図 7 に示す．TCP 通信でブロッキングモードを用いた場合はノンブロッキングモードの場合よりも 1.21 ms 高速であることが分かった．共有メモリを用いるとさらに 1.02 ms 高速になった．TCP 通信を用いた場合でも仮想ネットワーク経由での通信となるため，共有メモリを用いた場合と比べてそれほど大きな差にはならなかった．

データ受信処理のループにスリープを入れることにより，TCP 通信でノンブロッキングモードを用いる場合には取得時間が 360 μs 増加することが分かった．共有メモリを用いる場合にも 380 μs 増加した．これは IDS とエージェントの双方において最大で 200 μs 待たされるためである．しかし，スリープを入れない場合には CPU 使用率が 100%となる．スリープ時間と CPU 使用率の関係を測



(a) TCP 通信（ブロッキングモード） (b) TCP 通信（ノンブロッキングモード） (c) 共有メモリ

図 7: OS のバージョン情報の取得性能

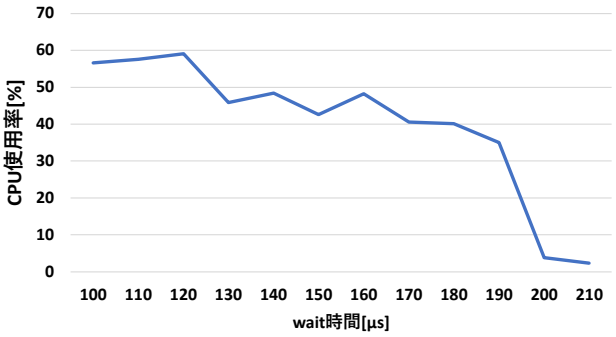


図 8: スリープ時間を変化させた場合の CPU 使用率

定した結果を図 8 に示す．この結果から，スリープ時間を 200 μs よりも短くすると CPU 使用率が急激に上昇することが分かる．

データの暗号化を行うことにより，TCP 通信でブロッキングモードを用いる場合には取得時間が 200 μs 増加することが分かった．ノンブロッキングモードの場合には 270 μs，共有メモリの場合には 350 μs 増加した．通信方法によって暗号化のオーバヘッドが異なる原因は不明であるが，データ受信処理のループ内のスリープの影響が考えられる．共有メモリを用いる場合に割り込みを用いることができれば，取得時間の増加を 200 μs 程度に抑えられる可能性がある．

5.2 OS のプロセス情報の取得

監視対象 VM から OS のプロセス一覧を取得する 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 と名前を取得して表示する．そのために，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．その結果，OS のプロセス一覧が表示されることを確認できた．

プロセス一覧の取得にかかった時間を図 9 に示す．TCP 通信でブロッキングモードを用いた場合はノンブロッキングモードの場合よりも 12.6%高速であることが分かった．共有メモリを用いるとさらに 39.5%高速になった．エージェントとの間でデータの送受信を繰り返す IDS の場合，TCP 通信で用いるモードの影響は小さくなり，共有メモリを用いる効果が大きくなることが分かった．

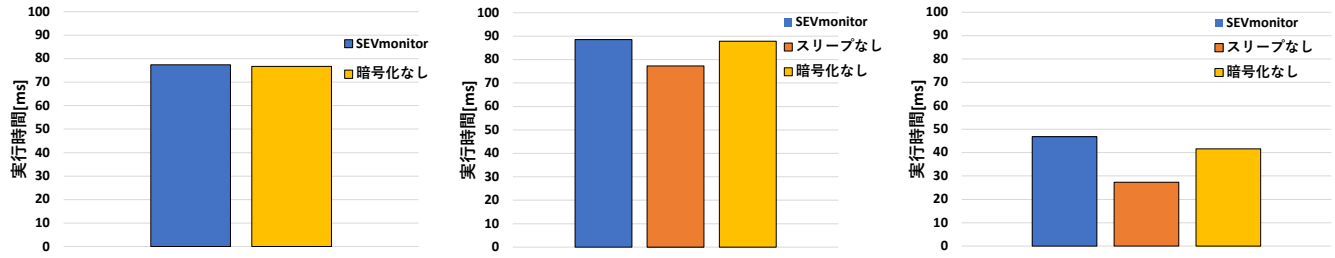
データ受信処理のループにスリープを入れることにより，TCP 通信でノンブロッキングモードを用いる場合には取得時間が 14.6%増加するだけで済むのに対し，共有メモリを用いる場合には 71.5%も増加した．増加する時間は共有メモリのほうが 8.2 ms 長いだけであるが，取得時間が短いために相対的な影響が大きい．

データの暗号化を行うことにより，TCP 通信ではいずれのモードでも取得時間が 740 μs だけ増加したが，共有メモリの場合には 5.2 ms 増加した．1 回だけメモリデータを要求した時には暗号化により取得時間が 200 μs 増加したため，TCP 通信で暗号化のオーバヘッドがこのように小さい原因は不明である．また，共有メモリの場合だけオーバヘッドが大きくなる原因についても調査する必要がある．

6. 関連研究

Intel Software Guard Extensions (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かす研究がいくつか行われている．SGX は AMD SEV とは異なる Trusted Execution Environment (TEE) であり，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 [8] や MigSGX [9] 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 [10] はサンドボックスを用いてエンクレイヴ内のサービスを安全に実行する．エンクレイヴ内にサンドボックスを構築するために Google Native Client (NaCl) [11] を用い，実行前のコード検査とランタイムチェックを行う．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．しかし，SEV を用いて保護された VM



(a) TCP 通信（ブロッキングモード）　(b) TCP 通信（ノンブロッキングモード）　(c) 共有メモリ

図 9: OS のプロセス一覧の取得性能

内のシステム全体に対して NaCl 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 [12] 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移動させる．エンクレイヴ外部に対しては安全な 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．それにより，Snort が攻撃されてもネットワークフローごとの情報などを盗まれないようにすることができる．SEC-IDS [13] は Snort 全体をエンクレイヴ内で動作させる．そのために，Graphene-SGX ライブラリ OS [14] を用いる．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 [15] 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 [16] は SCONE [17] 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．IDS が VM を監視できるようにするために，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．SMM は OS を含むユーザからアクセスできない独立した環境を提供する．HyperGuard[18] 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[19] 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[20] 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ため，様々な IDS やネットワークドライバを開発する労力が大きい．

一方，RemoteTrans[21] 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．メモリだけでなく，VM の仮想ディスクや仮想ネットワークを監視することもできる．クラウド内でエージェントを安全に実行するためにハイパーバイザを信頼する必要があるため，ハイパーバイザを攻撃されると取得するメモリデータを書き換えられてしまう恐れがある．

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 内でエージェントを安全に動作させ，IDS がエージェントと通信を行うことで VM のメモリデータを取得して監視を行う．IDS とエージェントは仮想ネットワークまたは共有メモリを用いて暗号化されたデータをやりとりする．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，取得性能を調べた．

今後の課題は，ivshmem の割り込みを利用できるようにして，データのやりとりの際のビジーループをなくすことである．また，SEV を用いて IDS VM のメモリを暗号化した場合に，共有メモリを用いてエージェントとの通信を行えるようにすることも必要である．他には，ネストした仮想化を用いたエージェントの実装を完了させ，OS から隠されたエージェントの実装方法についても検討を行う予定である．

謝辞

本研究成果は，国立研究開発法人情報通信研究機構の委託研究により得られたものです．

参考文献

[1] Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Instrusion Detection, *In Proceedings of Network and Distributed Systems Se-*

curity Symposium, pp. 191–206 (2003).
[2] 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/10threats2021.html⟩.
[3] Ben-Yehuda, M., Day, M., Dubitzky, Z., Factor, M., N.Har'El, Gordon, A., Liguori, A., Wasserman, O., Yassour, B.：Turtle Project: Design and Implementation of Nested Virtualization, *In Proceedings of USENIX Symp. Operating Systems Design and Implementation*, pp. 423–436 (2010).
[4] 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 Proceedings of International Conference on Virtual Execution Environment*, pp. 121–135 (2009).
[5] The QEMU project developers: Inter-VM Shared Memory device, (online), available from ⟨https://qemu-project.gitlab.io/qemu/system/ivshmem.html⟩.
[6] Macdonell, C.: ivshmem-uio, (online), available from ⟨https://github.com/shawnanastasio/ivshmem-uio⟩.
[7] Ozaki, Y., Kanamoto, S., Yamamoto, H. and Kourai, K.: Detecting System Failures with GPUs and LLVM, *In Proceedings of Asia Pacific Workshop on Systems* (2019).
[8] 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 Proceedings of IEEE International Conference on Dependable Systems and Networks* (2017).
[9] 中島健児，光来健一：Intel SGX を利用するコンテナのマイグレーション機構，コンピュータセキュリティシンポジウム (2018).
[10] Tyler, H., Zhiting, Z., X. Yuanzhong, P. S. and Emmett, W.: Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data, *In Proceedings of USENIX Symposium on Operating Systems Design and Implementation* (2016).
[11] Sehr, D., Muth, R., Biffle, C., Khimenko, V., Pasko, E., Schimpf, K., Yee, B. and Chen, B.: Adapting software fault isolation to contemporary cpu architectures, *In Proceedings of n USENIX Security* (2010).
[12] Shih, M., Kumar, M., Kim, T. and Gavrilovska, A.: Securing NFV States by Using SGX, *In Proceedings of ACM International Workshop on Security in Software Defined Networks and Network Function Virtualization* (2016).
[13] Kuvaiskii, D., Chakrabarti, S. and Vij, M.: Intrusion Detection System with Intel Software Guard Extension, *arXiv:1802.00508* (2018).
[14] Tsai, C., Porter, D. and Vij, M.: Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX, *In Proceedings of USENIX Annual Technical Conference* (2017).
[15] 中野智晴，光来健一：Intel SGX を用いた VM のメモリとディスクを安全な監視，コンピュータセキュリティシンポジウム (2019).
[16] 河村拓実，光来健一：Intel SGX と SCONE を用いた既存 IDS の安全なオフロード，第 149 回 OS 研究会 (2020).
[17] 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 Proceedings of the 12th USENIX Symposium on Operating System Design and Implementation*, pp. 689–705 (2016).
[18] Rutkowska, J. and Wojtczuk, R.: Preventing and detecting Xen hypervisor subversions, *Blackhat Briefings USA* (2008).
[19] Azab, A., Ning, P., Wang, Z., Jiang, X., Zhang, X. and Skalsky, N.: HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity, *In Proceedings of Conference on Computer and Communications Security*, pp. 38–49 (2010).
[20] Wang, J., Stavrou, A. and Ghosh, A.: HyperCheck: A hardware-assisted Integrity Monitor, *In Proceedings of International Symposium on Recent Advances in Intrusion Detection*, pp. 157–177 (2010).
[21] Kourai, K. and Juda, K.: Secure Offloading of Legacy IDSes Using Remote VM Introspection in Semi-trusted Clouds, *In Proceedings of the 9th IEEE International Conference on Cloud Computing*, pp. 43–50 (2016).