SMIインジェクションとuBPFを用いた機密VMの柔軟なメモリ監視

山口 紘輝[1]　末永 道正[1]　光来 健一[1]

概要：クラウドでは，CPU の隔離実行環境を用いて VM を保護する機密 VM が提供されている．しかし，機密 VM に侵入されると復号された機密情報が攻撃者に漏洩する可能性があるため，機密 VM においても侵入検知システム（IDS）による監視が必要である．IDS を VM 外に分離すると VM 内の攻撃者から保護できる一方，メモリが暗号化された機密 VM の場合には OS データを直接取得することはできなくなる．VM 内にエージェントを配置する既存手法では，エージェントの安全性を高めるほど監視対象システムの実行時オーバヘッドが大きくなるという問題がある．本稿では，エージェントの保護とシステム性能を両立する uBPF-SV を提案する．uBPF-SV は監視を行う時だけエージェントを機密 VM の BIOS 内で実行し，システムマネジメントモード（SMM）を用いて安全に実行する．IDS は SMI インジェクションと呼ばれる機構を用いて機密 VM 内の SMM エージェントを呼び出す．SMM エージェントは IDS によってロードされた uBPF プログラムを安全に実行して，監視に必要な一連の OS データを IDS に返す．一度にすべての OS データを取得できない場合には一貫性を保って分割取得を行う．uBPF-SV を用いて実験を行い，uBPF による一括取得により OS データを高速に取得できることが分かった．また，監視によるシステム性能への影響は一般的に小さく抑えられることを確認した．

1. はじめに

クラウド上の仮想マシン（VM）は，組織の業務データや個人情報など，機密性の高い情報を扱う基盤として広く利用されている．その一方で，クラウドへの侵入や内部不正によって，VM のメモリから情報を読み出されたり，実行状態を改変されたりする危険がある．こうした脅威に対し，Amazon Web Services，Google Cloud，Microsoft Azure をはじめとするクラウドでは，VM のメモリや実行状態を VM 外部から保護する機密 VM が提供されている．機密 VM は AMD SEV [1] や Intel TDX [2] など，CPU が提供する隔離実行環境（TEE）を用いて VM を保護する．

ただし，機密 VM が提供する保護は，VM 内部への侵入を検出したり，侵入後の不正操作を防いだりするものではない．機密 VM 内のシステムの実行中にはデータが復号されて扱われるため，機密 VM に侵入した攻撃者は通常の VM と同様に機密情報へアクセスできる．このため，機密 VM でも侵入検知システム（IDS）による継続的な監視が必要である．IDS を監視対象 VM 内に配置すると侵入者に停止または改変される可能性があることから，IDS を VM 外へ分離する IDS オフロードが用いられてきた [3]．しかし，機密 VM では外部からメモリを直接観測できず，従来の VM イントロスペクション [4] をそのまま適用できない．機密 VM 内のエージェントを介してメモリ上の OS データを取得する手法が提案されている [5] が，エージェントの隔離を強めるほど，監視対象システムの実行時オーバヘッドも大きくなる．

本稿では，エージェントの高い安全性と監視対象システムの低い実行時オーバヘッドを両立させる uBPF-SV を提案する．uBPF-SV はエージェントを機密 VM 内の BIOS に組み込み，CPU のシステムマネジメントモード（SMM）を用いて安全に実行する．別の機密 VM 内で動作する IDS は SMI インジェクションと呼ばれる機構を用いて SMM エージェントを呼び出す．IDS は SMM エージェントとの間の共有メモリに要求を格納した後，監視対象 VM にシステムマネジメント割り込み（SMI）を挿入する．呼び出された SMM エージェントはシステムメモリから取得した OS データを共有メモリに格納して IDS に返す．SMM エージェントには SMM 以外の CPU モードではアクセスできないため，侵入者から SMM エージェントを保護できる．また，SMM エージェントの実行中以外は通常のシステムと変わらないため，実行時オーバヘッドが生じない．

SMI インジェクションのオーバヘッドは大きいため，uBPF-SV は BIOS に uBPF プログラムを送り込むことに

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

ⓒ2026 Information Processing Society of Japan

よって OS データをまとめて取得する．uBPF プログラムのロード時に SMM エージェントが検証を行うことにより，BIOS 内で安全に実行することができる．OS データのあるシステムメモリや共有メモリには，SMM エージェントがヘルパー関数を用意して安全にアクセスする．1 回の SMI インジェクションで複数の uBPF プログラムを実行して複数種類の OS データを取得することもできる．逆に，可変長のプロセスリストなど取得した OS データが共有メモリに入り切らない場合は，複数回に分けて取得する．その際には，分割取得中に OS データの一貫性が失われていないか確認する．

uBPF-SV を OVMF [6], QEMU-KVM [7], および uBPF ランタイム [8] に実装し，SEV で保護された機密 VM を用いて実験を行った．その結果，proc ファイルシステムが提供するシステム情報を正しく取得できることを確認した．複数の uBPF プログラムを一括実行した場合，個別に実行する場合と比べて所要時間を 19.8%削減できた．また，システム性能への影響は高負荷な監視を高頻度で行わなければ十分に小さいことが分かった．

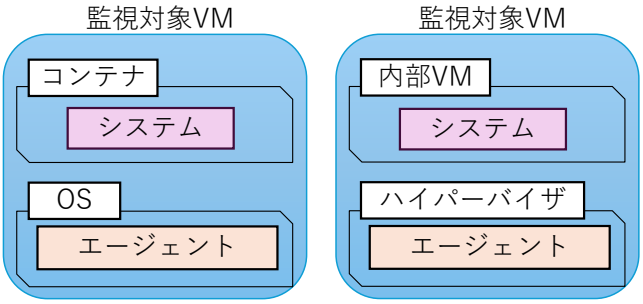
以下，2 章では機密 VM の監視における課題と，先行研究におけるエージェント配置のトレードオフについて述べる．3 章で uBPF-SV の設計を示し，4 章で SEV への対応を含む実装を説明する．5 章で動作確認と実験結果について述べる．6 章で関連研究を示し，7 章で本稿をまとめる．

2. 機密 VM の監視

機密 VM は，CPU によって提供される TEE を用いて VM 外部との境界を保護することができる点が通常の VM との大きな違いである．この保護境界はクラウドからの観測を制限するが，機密 VM 内のシステムの安全性を保証するものではない．システムが侵害されれば，攻撃者は機密 VM 内で復号されたデータを盗聴したり，ゲスト OS の権限を悪用したりすることができる．したがって，機密 VM においてもプロセス，メモリ使用量，ネットワーク接続などの状態を取得し，攻撃の兆候を検出する侵入検知システム（IDS）が必要となる．

IDS を監視対象 VM から分離する IDS オフロードには，攻撃者に機密 VM 内の管理者権限を奪われた場合でも IDS そのものを停止されにくいという利点がある．通常の VM に対しては，VM イントロスペクションを用いて VM 外部からメモリデータを読み出し，OS データを取得して監視を行うことができる．しかし，機密 VM では読み出したメモリデータが暗号化されていたり，読み出しが制限されていたりするため，従来と同じ手法では OS データを取得することができない．そのため，IDS を VM 内の攻撃者から分離することと，機密 VM の内部状態を観測することの二つの要件を同時に満たす必要がある．

SEVmonitor [5] は機密 VM 内にエージェントを配置し，



(a) コンテナを用いる構成　(b) 内部 VM を用いる構成

図 1: 監視対象とエージェントの配置例

VM 外の 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り，機密 VM の監視を可能にしている．エージェントは機密 VM 内で復号されたメモリを参照できるため，メモリ暗号化によって生じる観測上の制約の影響を受けない．IDS は別の機密 VM で実行することにより，クラウドからの攻撃を防ぐ．IDS とエージェントの通信には仮想ネットワークまたは共有メモリを用い，送信データを暗号化して保護する．

一方，機密 VM 内の監視対象システムにエージェントを配置すると，システムを侵害した攻撃者からエージェントを保護することができない．そのため，監視対象システムとエージェントの間に追加の隔離境界を設ける必要がある．エージェントをどの層へ配置するかによって，隔離の強さや実行時オーバヘッドなどにトレードオフがある．SEVmonitor における二つの配置を図 1 に示す．いずれも監視対象システムを隔離し，その外側にエージェントを配置する．これにより，監視対象システムが侵害された場合でも，隔離境界を越えてエージェントへ到達しにくくすることができる．

図 1(a) では，監視対象システムを機密 VM 内のコンテナに隔離し，エージェントをカーネル内に配置する．コンテナは同一カーネルを共有するため，機密 VM 内での追加の仮想化に伴うオーバヘッドを抑えやすい．しかし，コンテナの隔離が破られてカーネルを攻撃されると，エージェントも改変される可能性がある．また，コンテナに強い権限を与えるほど，エージェントが攻撃される危険は大きくなる．したがって，低いオーバヘッドと引き換えに，監視対象システムで利用できる機能とエージェントの保護範囲が制約される．

図 1(b) では，監視対象システムを機密 VM 内の内部 VM に隔離し，エージェントをハイパーバイザ内に配置する．VM 境界による隔離を利用できるため，コンテナを用いる構成よりも監視対象システムに強い権限を与えやすい．その反面，機密 VM 内で VM を実行するには Nested SEV [9] が必要となり，監視を行っていない通常時の処理にも仮想化および SEV のコストが二重に加わる．このように，監

ⓒ2026 Information Processing Society of Japan

視対象システムが利用できる機能に制限がなく，エージェントも安全に実行できるが，オーバヘッドが大きい．

3. uBPF-SV

3.1 脅威モデル

機密 VM の作成に用いられる TEE は信頼できるものとし，脆弱性はないものとする．また，CPU のシステムマネジメントモード（SMM）も信頼でき，脆弱性は仮定しない．監視対象の機密 VM の BIOS 内で動作する SMM エージェントおよび，別の機密 VM 内で動作する IDS は信頼できるものとする．機密 VM の BIOS は起動時にクラウドが変更することができるが，リモートアテステーションを行うことによって改ざんされていないことを検知することができる．機密 VM のゲスト OS もリモートアテステーションによって改ざんを検知でき，起動中には攻撃を受けないものとする．また，IDS と SMM エージェントが共有するルート鍵は安全に事前共有され，攻撃者には漏えいしないものとする．

一方，クラウドの内部犯やクラウドへの侵入者を想定し，ハイパーバイザは信頼できないものとする．監視対象 VM への侵入者も想定するが，攻撃者はゲスト OS とハイパーバイザの両方の攻撃には成功しないものとする．また，VM 内の侵入者とクラウドは結託しないものとする．IDS と SMM エージェント間の共有メモリは機密 VM およびハイパーバイザとも共有されるため，監視対象 VM への侵入者やハイパーバイザが盗聴や改ざんを行えるものとする．そのため，攻撃者は共有メモリ上の要求や制御応答，取得した OS データなどの改ざんおよびリプレイ攻撃を試みることができる．

3.2 SMM エージェントを用いた監視

uBPF-SV は，機密 VM の BIOS 内で SMM を用いてエージェントを動作させ，OS データを取得する．SMM は BIOS が使用する x86 プロセッサの動作モードであり，BIOS が管理する専用メモリ領域（SMRAM）を用いて，通常の VM 実行環境から分離された実行環境を提供する．仮想 CPU においても SMM のアクセス制御がエミュレートされるため，SMM 以外のモードで動作する攻撃者のコードから，SMRAM 上の SMM エージェントのコードやデータにはアクセスできない．また，クラウドも機密 VM のメモリ上にある SMRAM にアクセスすることはできない．したがって，機密 VM 内に侵入した攻撃者やクラウドによる SMM エージェントの改ざんや無効化を防ぐことができる．また，SMM エージェントが実行されていない間はオーバヘッドが生じないため，エージェントの保護とシステム性能の維持を両立することができる．

uBPF-SV では，監視対象 VM の外部で動作する IDS が SMI インジェクションと呼ばれる機構を用いて BIOS 内の SMM エージェントを呼び出す．IDS はまず監視要求を SMM エージェントとの間に確立した共有メモリに書き込み，ハイパーバイザにシステムマネジメント割り込み（SMI）の挿入を要求する．ハイパーバイザはこの要求を受けて，監視対象 VM へ SMI を挿入する．これにより BIOS 内の SMI ハンドラが呼び出される．BIOS 内で共有メモリをポーリングする必要はなく，監視要求がないときには監視対象 VM の性能に影響を与えない．SMI ハンドラは共有メモリを検査し，有効な要求が存在する場合にのみ SMM エージェントを実行する．処理が完了すると，SMM エージェントは共有メモリに結果を書き込み，IDS が共有メモリから結果を取得する．

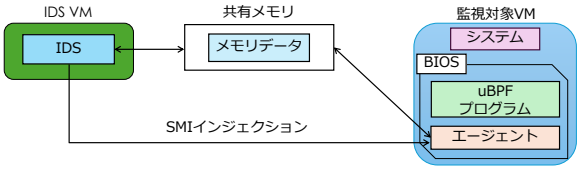


図 2: uBPF-SV のシステム構成

3.3 uBPF による OS データの取得

uBPF-SV は uBPF を用いて，柔軟かつ安全な OS データの取得を行う．VM 内の OS データを一つ取得するたびに SMI インジェクションを行うと，監視のオーバヘッドが大きくなる．しかし，SMM エージェントに複雑な OS データを取得するロジックを実装すると，柔軟性が低下し，脆弱性によって BIOS が攻撃を受ける可能性が高くなる．そこで uBPF-SV は，uBPF プログラムを IDS から BIOS へ送り込み，OS データをまとめて取得する．uBPF は，Linux カーネルで用いられる eBPF [10] のユーザ空間実装であり，プログラムを安全に実行することができる．uBPF を用いることにより，送り込んだプログラムが SMM エージェントや BIOS のメモリを破壊しないようにすることができる．例えば，プロセスリストのような OS のデータ構造をたどり，一連の OS データをまとめて取得できる．これにより，SMI インジェクションの回数を削減して，監視性能を向上させることができる．

SMM エージェントは uBPF プログラムのロード時に検証器を実行し，uBPF プログラムによるメモリアクセスに対して境界チェックを行う．そのため，uBPF プログラムはゲスト OS のシステムメモリや IDS との間の共有メモリにアクセスすることができない．これらのメモリへ制限された方法でアクセスするために，SMM エージェントは uBPF プログラムに対してヘルパー関数を提供する．主なヘルパー関数は，ゲスト OS の仮想アドレスを指定してデータを読み出す関数と，取得したデータを共有メモリへコピーする関数である．不正なメモリアクセスを防ぐために，ヘルパー関数はアドレス変換に失敗する仮想アドレス

ⓒ2026 Information Processing Society of Japan

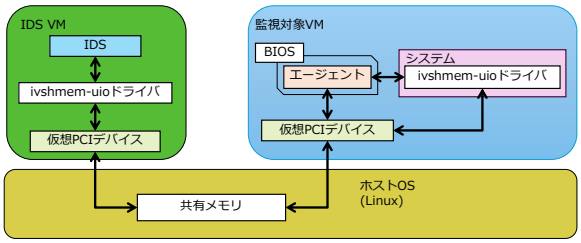


図 3: 共有メモリの構成

へのアクセスを拒否し，共有メモリの正しい範囲内へのコピーのみを許可する．

SMM エージェントは SMI インジェクションの際に，複数の uBPF プログラムを実行することにより，複数の種類の OS データもまとめて取得することができる．これにより，さらに SMI インジェクションの回数を削減することができる．一方，プロセス一覧のような可変長データは，結果を格納する共有メモリに収まらない場合がある．その場合には，SMI インジェクションの回数は増えるが，IDS は複数回に分けて結果を取得する．SMM エージェントの実行中はゲスト OS の実行が停止するが，実行と実行の間はゲスト OS が動作する．そのため，取得する OS データが変化し，重複や欠落が発生する可能性がある．そこで，IDS 側で OS データの一貫性を確認し，一貫性が保たれていなければ取得し直す．

4. 実装

uBPF-SV を OVMF [6], QEMU-KVM [7], および uBPF ランタイム [8] に実装した．また，監視対象 VM に共有メモリを扱うためのカーネルモジュールを実装した．

4.1 共有メモリ

IDS と SMM エージェントの通信に用いる共有メモリは QEMU の ivshmem を用いて作成する．図 3 に示すように，ivshmem はホスト上の同一メモリ領域を監視対象 VM と IDS VM へ仮想 PCI デバイスとして提供する．OVMF にはこの仮想 PCI デバイスのドライバが存在しないため，代わりに監視対象 VM のカーネルモジュールが PCI BAR から共有メモリの物理アドレスとサイズを取得し，SMM エージェントに登録する．IDS を実行する機密 VM である IDS VM では共有メモリを UIO デバイスとして提供し，IDS プロセスのアドレス空間へマップする．

共有メモリ上には，メールボックス，暗号化メッセージ用バッファ，および uBPF プログラムごとの結果領域を配置する．メールボックスは要求や応答の内容そのものを格納する領域ではなく，両者の受け渡しを制御するための領域である．メールボックスには，要求および応答が有効であることを示すフラグ，シーケンス番号，暗号化メッセージ用バッファの物理アドレスとデータ長，ノンス，認証タグ，および処理状態を格納する．

暗号化メッセージ用バッファは要求と応答で共用する．要求時には，コマンド種別，プログラム ID，結果領域の位置とサイズなどを含む要求ペイロードを作成する．ロード要求の場合は uBPF バイトコードも要求ペイロードに含める．IDS は要求ペイロードを暗号化してメッセージ用バッファへ書き込み，その物理アドレス，要求ペイロードのサイズ，ノンス，認証タグなどをメールボックスに設定する．SMM エージェントはメールボックスを参照して要求ペイロードを読み出し，復号して要求を処理する．

要求の処理後，SMM エージェントは同じメッセージ用バッファを制御応答ペイロードで上書きする．制御応答には，要求の処理が成功したかを示す状態，処理したコマンド，完了コード，uBPF プログラムの返り値，およびエラーの詳細を格納する．これは要求の処理結果を IDS へ通知するための応答であり，取得した OS データ本体ではない．uBPF プログラムが取得したプロセス情報などの監視結果は，プログラムごとに割り当てた別の結果領域へ書き込む．IDS は制御応答を確認した後，応答に含まれるノンスと認証タグを用いて結果領域を復号・検証し，処理が成功していれば監視結果を利用する．

共有メモリはハイパーバイザからも参照できてしまうため，要求ペイロード，制御応答ペイロード，および監視結果本体を AES-256-GCM で暗号化・認証する．IDS と SMM エージェント間で事前に共有したルート鍵と要求ごとに増加するシーケンス番号から HKDF-SHA256 を用いて，要求・応答用と結果用の鍵を分離して導出する．要求にはランダムなノンスを用い，応答と監視結果にはシーケンス番号から別々に導出したノンスを用いる．メールボックスの主要なメタデータに加え，結果領域の位置と長さを付加認証データに含めることで，暗号文と要求情報の改ざんを検出する．また，SMM エージェントは処理済みのシーケンス番号を記録し，同じ要求の再実行を拒否する．

4.2 SMI インジェクション

SMI インジェクションの実装を図 4 に示す．IDS は uBPF プログラムのロード要求や実行要求を共有メモリに書き込んだ後，監視対象 VM の QEMU 内部で動作する QMP サーバへ TCP/IP で接続し，JSON 形式の `inject-smi` コマンドを送信する．QEMU は KVM へ SMI の挿入を要求し，KVM は監視対象 VM の仮想 CPU に SMI を挿入する．これにより，監視対象 VM の OVMF 内の SMI ハンドラが呼び出される．

4.3 SMM エージェント

4.3.1 ゲスト OS による初期化

監視対象 VM のカーネルモジュールは共有メモリに初期化要求を暗号化して格納し，I/O ポート `0xB2` への書き

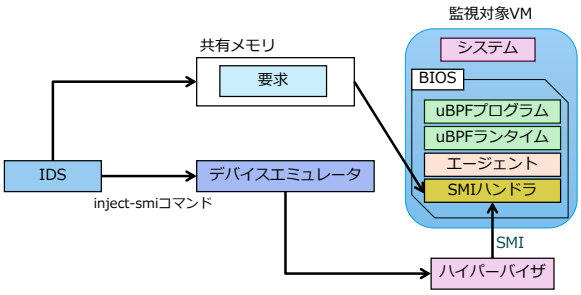


図 4: SMI インジェクション

込みを行って SMI を発生させる．通常の VM の場合，レジスタに共有メモリの物理アドレスを格納しておくと，ハイパーバイザがレジスタの値を BIOS のメモリに保存し，それを SMM エージェントが受け取ることができる．しかし，SEV によって保護された機密 VM の場合，ハイパーバイザが BIOS のメモリに保存したレジスタの値は暗号化されないため，VM 内の SMM エージェントは正常に読み込むことができない．

そこで，機密 VM の固定物理アドレス `0x001FF000` に初期メールボックスを配置する．初期メールボックスには，共有メモリ上に格納した暗号化済み初期化要求の物理アドレスとサイズを格納する．初期化要求ペイロードには，共有メモリ領域のベース物理アドレスとサイズを格納する．SMM エージェントは固定物理アドレスから初期メールボックスを読み出し，要求ペイロードを復号した後，共有メモリの物理アドレスとサイズを取得する．

同様に，SMM エージェントは SMI が挿入された時の CR3 レジスタの値を SMRAM から読み込むことによって，ページテーブルの物理アドレスを受け取ることができない．そこで，監視対象 VM 内のカーネルモジュールがゲスト OS のページテーブル情報を取得し，初期メールボックスに格納することによって初期化時に SMM エージェントへ通知する．カーネルモジュールはページテーブルの仮想アドレスを実行中のプロセスの構造体から取得し，それを物理アドレスに変換する．

また，このカーネルモジュールは `init_task` の実行時アドレスも初期メールボックスに格納し，SMM モジュールに通知する．機密 VM のカーネルにおいてアドレス空間配置のランダム化が行われると，カーネルシンボルのビルド時のアドレスと実行時のアドレスが一致しない．SMM エージェントは uBPF バイトコードのロード時に uBPF バイトコード内のカーネルアドレスを補正するために，通知された実行時アドレスを基準として用いる．

SMM エージェントは初期化要求を検証した後，共有メモリを SMM のページテーブルに登録する．SEV はページテーブルエントリ（PTE）の C-bit でページの暗号化を制御するが，共有メモリは SEV によって暗号化されないため，PTE には C-bit をセットせずに登録する．

4.3.2 IDS からの要求の処理

ロード要求は uBPF バイトコード，プログラム ID，結果領域のオフセットとサイズからなり，IDS が暗号化して共有メモリ上のメッセージ用バッファに格納する．IDS は要求の位置や長さなどをメールボックスに設定する．SMM エージェントは uBPF ランタイムを呼び出すことによって uBPF バイトコードをロードし，uBPF VM を作成する．ロード済み uBPF プログラムを管理するために，16 個のプログラムスロットが用意されている．各スロットはプログラム ID，uBPF VM，JIT コンパイル後のコード，uBPF バイトコードのハッシュ値，および結果領域のオフセットとサイズを保持する．これにより，実行要求のたびに検証と JIT コンパイルのコストを負担せずに済む．同じプログラム ID に同じ長さとハッシュ値を持つ uBPF バイトコードが再度ロードされた場合は，作成済みの uBPF VM と JIT コンパイル後のコードを再利用する．

実行要求は 1 つ以上のロード済みプログラム ID，結果領域のオフセットとサイズからなり，ロード要求と同様に暗号化メッセージ用バッファを用いて送信する．SMM エージェントは指定された uBPF プログラムを順に実行し，各プログラムに割り当てられた別々の結果領域へ出力する．その際に，指定された結果領域がロード時に登録した領域と一致することを確認してから，JIT コンパイル後のコードを実行する．実行後は状態コードを含む制御応答を暗号化メッセージ用バッファへ書き込み，メールボックスの応答完了フラグを設定する．

4.4 uBPF ランタイム

uBPF ランタイムを SMM エージェント内で実行できるようにするために，標準 C ライブラリに依存する型，メモリ管理，文字列操作，乱数生成などを OVMF の API で置き換えた．uBPF ランタイムはシステムメモリの読み出しや共有メモリへの書き込みを行うヘルパー関数を関数番号と対応づけて登録する．uBPF バイトコードは `call` 命令でこの関数番号を指定することにより，ヘルパー関数を呼び出す．

uBPF バイトコードをロードする際には命令列とメモリアクセスを検証し，検証に失敗したバイトコードのロードは拒否する．検証に成功したバイトコードは JIT コンパイラを用いて x86-64 のネイティブコードへ変換する．生成コード用のページを確保してコードを配置した後，コードを実行できるように SMM Memory Attribute Protocol を用いて実行不可属性を解除する．

uBPF バイトコードは OS データを解析するために，システムメモリを参照するヘルパー関数を用いてゲスト OS の仮想アドレスに対応するデータを読み出す．このヘルパー関数は，ゲスト OS のページテーブルを用いて仮想アドレスを物理アドレスへ変換する．その際に，ゲスト OS

ⓒ2026 Information Processing Society of Japan

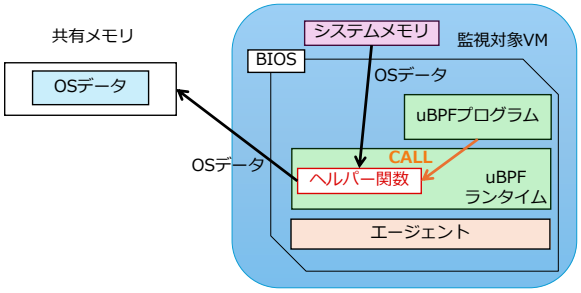


図 5: ヘルパー関数を用いたメモリコピー

の PTE には C-bit が含まれるため，C-bit をマスクすることでゲスト物理アドレスを取得する．そして，取得したアドレスに対応するページを一時的にマップすることで，4 段のページテーブルをたどる．

アドレス変換に成功した場合は物理ページを SMM のページテーブルに登録する．システムメモリのページの PTE には C-bit がセットされているため，SMM のページテーブルの PTE にも C-bit をセットすることにより，システムメモリを復号された状態で読み出せるようにする．そして，マップしたシステムメモリから必要なデータだけを SMRAM 内の一時領域へコピーして uBPF バイトコードに渡す．

uBPF バイトコードは共有メモリへコピーするヘルパー関数を用いて OS データを出力する．このヘルパー関数には書き込み先を共有メモリの先頭からのオフセットで指定する．uBPF ランタイムはオフセットとデータサイズが共有メモリの範囲内にあり，かつ，実行中の uBPF バイトコードに割り当てた結果領域内に収まることを確認してから，SMRAM 内の一時領域にデータを書き込む．SMM エージェントは実行完了後に実際にデータが格納された範囲を AES-256-GCM で暗号化し，暗号文のみを共有メモリの結果領域へ書き込む．末尾の未使用領域は暗号化対象から除外し，IDS 側でゼロとして復元する．

4.5 LLView による uBPF プログラムの生成

OS データを取得する uBPF プログラムの生成には LLView [11] を用いた．LLView は監視処理を Clang で LLVM bitcode へ変換した後，二つの LLVM pass を適用する．一つ目の pass は，Linux カーネルの `System.map` を参照し，大域変数をビルド時の仮想アドレスへ置き換える．二つ目の pass は，カーネルデータへの load 命令を，システムメモリを SMRAM 内の一時領域へコピーするヘルパー関数の呼び出しへ置き換える．変換後の bitcode を eBPF オブジェクトへコンパイルし，`.text` セクションからバイトコードを取り出す．

4.6 OS データの分割取得

共有メモリには uBPF プログラムごとに固定長の結果領域を割り当てる．そのため，プロセスや TCP 接続などの数が多い場合は，一度の実行要求ですべての結果を格納できない．そこで，プロセス，TTY，TCP 接続の一覧については，実行要求に取得開始位置と一回あたりの最大取得件数を含める．SMM エージェントは指定された範囲を取得し，後続データが存在する場合は結果領域に継続フラグを設定する．継続フラグを受け取った IDS は，取得件数だけ開始位置を進めて実行要求を繰り返す．複数の uBPF プログラムを同時に実行する場合は，取得が完了したプログラム ID を次の実行要求から除外する．

例えば，プロセスリストを分割取得する場合，SMM エージェントは実行要求のたびに `init_task` からリストをたどり直し，IDS が指定した件数を読み飛ばして取得を再開する．隣接する取得単位の連続性を確認するため，直前の取得単位の最後のプロセスを，次の取得単位の先頭でも重複して取得する．IDS は両者のプロセス ID が一致することを確認し，一致した場合は重複分を除外して結果を連結する．一致しない場合はすでに情報を取得したプロセスが終了したということであるため，プロセスリストの一貫性を保つためにそれまでの結果を破棄して先頭から取得し直す．

5. セキュリティ分析

uBPF-SV の SMM エージェントは VM の外側と内側の両方の攻撃から保護される．機密 VM のメモリや CPU レジスタは TEE による暗号化やアクセス制御によって保護されるため，VM の外側のハイパーバイザ等が盗聴や改ざんを行うことはできない．ただし，機密 VM の BIOS はクラウドによって提供されるため，BIOS イメージに含まれる SMM エージェントは VM の起動前にクラウドによって改ざんされる恐れがある．この攻撃については，VM 起動時にリモートアテステーションを行うことによって検知することができる．

VM 内の侵入者からの攻撃に対しては，ハイパーバイザによる SMM エミュレーションによって防ぐことができる．SMM エージェントが用いる SMRAM はハイパーバイザが VM に SMI ハンドラを実行させる時に VM のメモリにマッピングされ，SMI ハンドラの実行が終了するとマッピングが解除される．このように，ハイパーバイザが SMRAM のマッピングを制御しているため，VM 内では SMM での実行時以外にはアクセスすることができない．ハイパーバイザに SMRAM のマッピングを解除しないという攻撃を行われると，VM 内の侵入者は SMRAM にアクセスすることができるようになるが，uBPF-SV では，攻撃者はクラウドと VM の両方の攻撃には成功せず，VM 内の侵入者とクラウドは結託しないことを仮定している．

ハイパーバイザは IDS から要求された SMI を挿入しないことによって，SMM エージェントの呼び出しを妨害する攻撃が考えられる．IDS は SMI 挿入要求の送信後，QMP

ⓒ2026 Information Processing Society of Japan

表 1: 実験環境（ホスト）

項目	構成
CPU	AMD EPYC 7443P
メモリ	256 GiB
OS	Linux 6.8
仮想化ソフトウェア	QEMU 8.1.50

表 2: 実験環境（VM）

項目	IDS VM	監視対象 VM
仮想 CPU	1	1
メモリ	4 GiB	4 GiB
OS	Linux 6.8	Linux 6.8
SEV	無効	有効

応答と，送信した要求と同じシーケンス番号を持つメールボックス応答を待つ．一定時間内に QMP 応答を受信できない場合や，応答準備完了フラグが設定された対応するメールボックス応答を受信できない場合，IDS は監視要求が完了しなかったことをタイムアウトとして検知することができる．また，応答ペイロードは AES-256-GCM によって認証され，シーケンス番号も検査されるため，ハイパーバイザが過去の応答を再送して要求が完了したように見せかけることはできない．ただし，IDS はこのような攻撃を検知できるだけであり，SMM エージェントの実行を強制することや，意図的な妨害と通信障害を区別することはできない．

6. 実験

uBPF-SV の動作を確認し，監視性能およびシステム性能への影響を調べる実験を行った．実験に用いたホストと VM の構成をそれぞれ表 1 と表 2 に示す．監視対象 VM では SEV を有効にし，BIOS として SMM エージェントを組み込んだ OVMF を用いた．共有メモリの容量は 1 MB であった．特に断らない限り，30 回測定を行い，平均と標準偏差を求めた．

6.1 動作確認

Linux の proc ファイルシステムによって提供される情報をそれぞれ取得する uBPF プログラムを用いて，監視対象 VM のシステム情報を取得する IDS を実行した．この IDS は表 3 に示す項目それぞれについて監視を行い，監視対象 VM 内で取得した値と一致することを確認した．`plist` は `comm` と似ているが，より多くの 581 プロセスに対してプロセス ID とプロセス名を監視し，5 回に分割して取得できることが確認できた．その際に，プロセスリストの変化を検出する一貫性確認を行っても再試行は発生しなかった．

次に，分割取得中にプロセスリストを変化させ，境界検査が不一致を検出できるかを確認した．300 個の `sleep` プロセスを追加した後，IDS がプロセス一覧を分割取得している間に，プロセスの終了と生成を 1,200 回繰り返した．この条件で IDS を 10 回実行した結果，全実行で隣接する取得単位の境界の不一致を検出した．

表 3: 動作を確認した監視項目

監視項目	取得結果
`pid_max`	プロセス ID の最大値を取得
`osrelease`	カーネルバージョンを取得
`stat`	システムの統計情報を取得
`meminfo`	メモリの利用状況を取得
`tty_drivers`	7 つの TTY ドライバの情報を取得
`tcp`	2 つの TCP 接続情報を取得
`comm`	138 プロセスの名前を 2 回で取得
`status`	138 プロセスの状態を 2 回で取得
`plist`	581 プロセスの情報を 5 回で取得

6.2 SMI インジェクションにかかる時間

uBPF プログラムを実行しない NOOP 要求を用いて，SMI インジェクションを含む要求処理時間の内訳を測定した．要求と応答を AES-256-GCM で保護する場合としない場合について各 30 回測定した．IDS 側では，要求生成，暗号化またはコピー，QMP 往復，QMP 応答後のメールボックス完了待ち，応答の復号またはコピー，および要求全体を個別に計測した．暗号化時間には OpenSSL の初期化，HKDF，ノンス生成，および AES-GCM の処理が含まれる．

結果を図 6 に示す．AES-256-GCM で保護する場合，QEMU 内で SMI 挿入処理を呼び出すのにかかる時間は 153 μs であり，QMP 応答後の完了待ちは 332 μs であった．よって，SMI 挿入から SMM エージェントの完了までに少なくとも 485 μs かかることが分かった．実際には，さらに QEMU が QMP 応答を生成してから IDS が受信するまでの時間がかかる．SMM エージェントの実行にかかる時間は 224 μs であった．IDS での要求生成は 269 μs，要求暗号化は 1,425 μs，応答復号は 13 μs であり，要求全体では 2,426 μs であった．一方，AES-256-GCM を用いない場合，要求全体では 709 μs かかり，暗号化による増加分 1,717 μs の大部分は，IDS 側の要求暗号化の差 1,421 μs で説明できる．

6.3 監視性能

6.3.1 OS データの取得時間

IDS が uBPF プログラムをロードした後，実行要求を送信してから制御応答と監視結果の復号・検証が完了するまでの時間を測定した．図 7 に実行時間を示す．ただし，実行時間には uBPF プログラムのロード時間は含ま

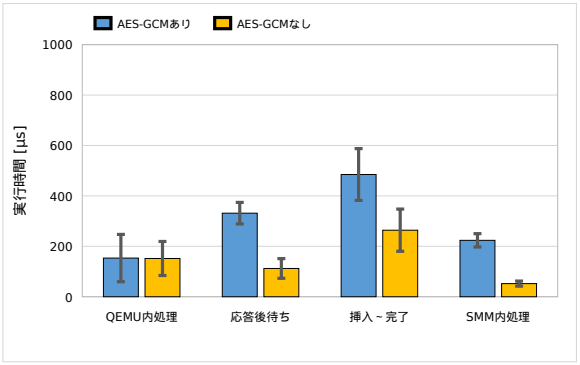


図 6: SMI インジェクションを含む NOOP 要求の内訳

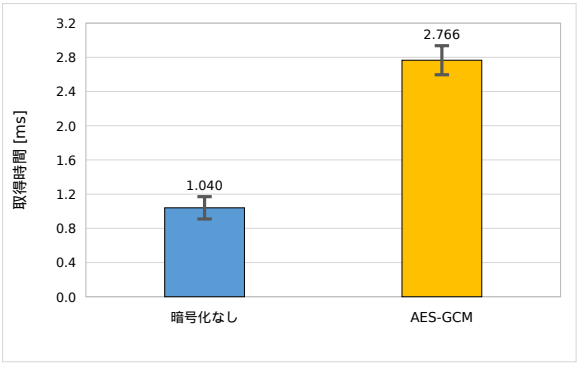


図 8: 監視結果の暗号化による取得時間の変化

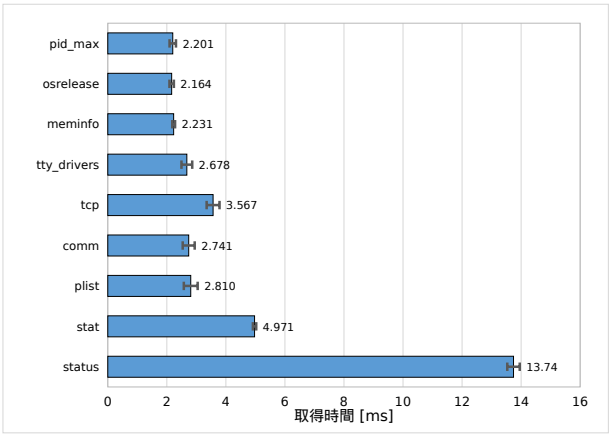


図 7: OS データの取得時間

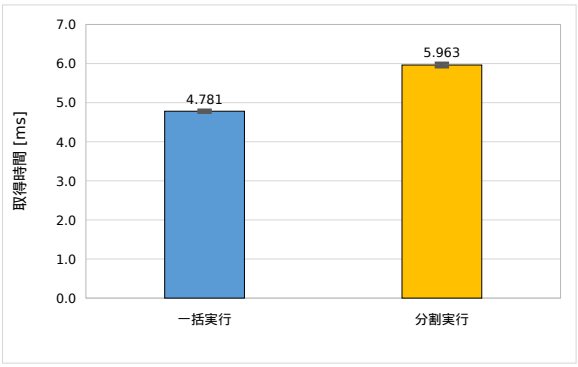


図 9: 複数 uBPF プログラムの一括実行と分割実行の比較

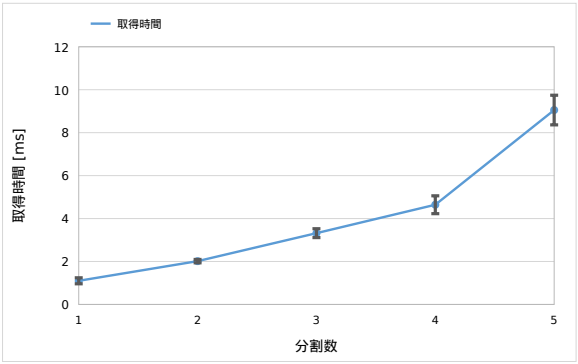


図 10: プロセス一覧の分割取得性能

れない．要求，制御応答，および使用済みの結果領域は AES-256-GCM で保護した．

取得時間は，`pid_max`，`osrelease`，`meminfo`，`tty_drivers` で平均 2.2–2.7 ms であった．プロセス ID とプロセス名を取得する `plist` は平均 2.8 ms，プロセスごとに多数の情報を取得する `status` は平均 13.7 ms を要した．暗号化するデータ量は監視項目によって異なり，`pid_max` の 147 B から `status` の 116 KB までであった．

監視結果を共有メモリ上で保護するためのオーバヘッドを調べた．116 プロセスの `plist` を 1 回の実行要求で取得し，要求，制御応答，および監視結果を AES-256-GCM で保護する場合と保護しない場合を各 30 回測定した．AES-256-GCM を用いる場合に暗号化した監視結果は 3.6 KB であった．結果を図 8 に示す．取得時間は保護によって平均 1.7 ms 増加した．

6.3.2 一括実行による高速化

`meminfo`，`tty_drivers`，`tcp` を取得する 3 つの uBPF プログラムを用いて，1 回の SMI インジェクションで一括実行する場合と，3 回の SMI インジェクションを行って 1 つずつ分割実行する場合を比較した．各 uBPF プログラムは事前にロードしておき，実行要求に要した時間の合計を測定した．結果を図 9 に示す．一括実行は平均 4.8 ms，分割実行は平均 6.0 ms であり，一括実行によって取得時間を 19.8%削減できることが分かった．これは SMI インジェクションに加え，暗号化した制御応答の受け渡し回数を 3 回から 1 回に削減できたためである．

次に，`plist` のプロセス数を変えて，プロセス一覧を 1 回から 5 回に分割して取得する場合について比較を行った．分割回数自体の影響を調べるため，この実験では監視結果本体を平文で受け渡す実験モードを用いた．1 回の取得上限を 128 件とし，隣接する取得単位の境界検査を行いながら，実行時間を測定した．結果を図 10 に示す．平均取得時間は，1 回から 4 回に分割された場合は，回数の増加に応じて取得時間がおおむね線形に増加した．一方，5 回に分割された場合は 1 回で取得できた場合の 8.2 倍の時間を要し，線形な増加よりも大きくなった．なお，いずれの条件でも境界不一致による再試行は発生しなかった．

ⓒ2026 Information Processing Society of Japan

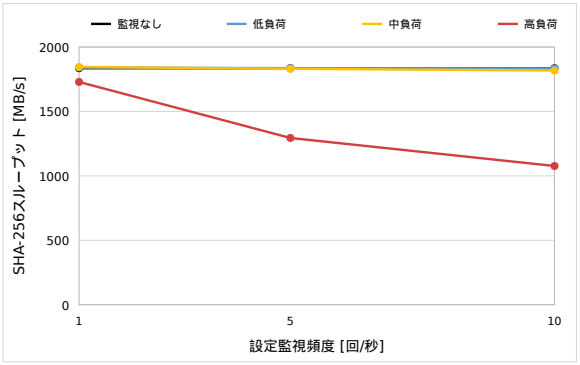


図 11: 監視中の SHA-256 スループット

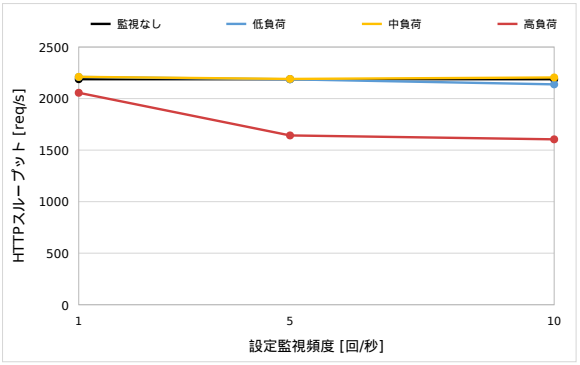


図 12: 監視中のウェブサーバのスループット

6.4 監視によるシステム性能への影響

6.4.1 SHA-256 ベンチマーク

監視対象 VM で OpenSSL の SHA-256 ベンチマークを実行し，監視によるシステム性能への影響を調べた．`meminfo` の実行要求を送る低負荷な監視，`plist` の実行要求を送る中程度の負荷の監視，および，これら 2 つを含む 5 つの uBPF プログラムの一括実行要求を送る高負荷な監視を行った．監視は 1，5，10 回/秒の頻度で行い，それぞれ 5 回ずつ測定した．比較として，監視を行わない場合についても測定を行った．

結果を図 11 に示す．10 回/秒に設定した低負荷および中程度の負荷の監視では，監視なしに対する低下率はそれぞれ 0.09%と 1.0%にとどまった．一方，高負荷の監視では，1，5，10 回/秒に設定したときの低下率はそれぞれ 5.8%，29.5%，41.3%であった．さらに，10 回/秒に設定した高負荷の監視の実効頻度は平均 6.7 回/秒にとどまった．これは，一つの監視要求が完了してから次の要求を送る現在の実装では，高負荷の監視処理時間が設定した 100 ms の周期に収まらなかったためである．したがって，多くの監視項目を一括実行する場合には，監視頻度を抑えるか，取得項目を絞る必要がある．

6.4.2 ウェブサーバ性能

監視対象 VM で Python 標準の HTTP サーバを動作させ，8 KiB の静的ファイルを配信した．IDS VM から並行度 8 で HTTP/1.0 接続を用いてスループットとレイテンシを測定した．6.4.1 節と同様の監視を行い，監視を行わない場合についても測定を行った．スループットを図 12 に示す．低負荷と中程度の負荷の監視では，10 回/秒でもスループットの変化は 2.3%以内であった．一方，高負荷な監視では，監視なしと比較して，1，5，10 回/秒でスループットがそれぞれ 6.0%，24.9%，26.7%低下した．これは，高負荷な監視では SMM エージェントの実行による監視対象 VM の停止時間が長くなり，HTTP 要求の処理を妨げるためである．

平均レイテンシを図 13 に示す．低負荷と中程度の負荷の監視では，監視頻度を上げても平均レイテンシの増加は小さかった．一方，高負荷な監視では，監視なしと比較して，1，5，10 回/秒で平均レイテンシがそれぞれ 4.4%，29.6%，32.0%増加した．また，5 回/秒以上では p99 レイテンシが約 31.7 ms に達した．このことから，高負荷な監視の影響は平均性能だけでなくテールレイテンシにも大きく現れることが分かる．

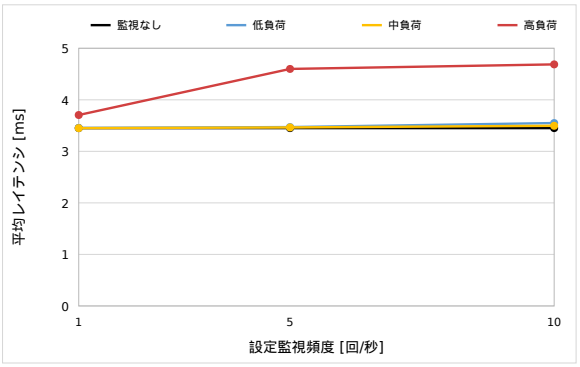


図 13: 監視中のウェブサーバのレイテンシ

7. 関連研究

これまでに，物理マシンにおいて OS やハイパーバイザを監視するために，BIOS 内で SMM を用いる様々な手法が提案されてきた．HyperGuard[12] は，SMM を用いてハイパーバイザのコード領域やページテーブルの整合性を検査する．SMI ハンドラはタイマ割り込みを用いて呼び出される．BIOS 内で IDS が動作するため，監視内容を変更する際には BIOS を更新する必要がある．それに対して，uBPF-SV は uBPF プログラムを送り込むことで様々な監視を行うことができる．

HyperCheck[13] は，BIOS 内の SMI ハンドラで NIC にアクセスしてメモリデータを転送し，リモートホストでシステムコールテーブルなどの整合性を検査する．SMI ハンドラを呼び出すための SMI は PCI NIC の MessageSignaled Interrupts（MSI）を用いて発生させる．リモートホストで様々な IDS を動作させることができるが，転送するメモリ領域は固定であるため，監視に必要なメモリ領域が変わる場合には BIOS の更新が必要になる．uBPF-SV では監視

ⓒ2026 Information Processing Society of Japan

対象 VM から独立した機密 VM で様々な監視を行うことができるのに加えて，監視に必要なデータも uBPF プログラムを用いて柔軟に取得することができる．

HyperSentry [14] は，SMM を用いてハイパーバイザにエージェントを挿入し，割り込みを禁止したり他の CPU コアを停止させたりすることでハイパーバイザ内でエージェントを安全に実行して整合性の検査を行う．SMI ハンドラを呼び出すために，リモートホストから Intelligent PlatformManagement Interface（IPMI）経由で Baseboard Management Controller（BMC）にアクセスして SMI を発生させる．監視結果は IPMI 経由でリモートホストに返される．SMI ハンドラの変更は基本的に必要となることはないが，エージェントを変更する際には BIOS の更新が必要となる．

SSdetector [15] は，SMM と Intel SGX [16] を組み合わせることで安全かつ柔軟に IDS を実行することを可能にしている．アプリケーション内に作成される保護領域である SGX エンクレイヴ内で IDS を実行し，SMM プログラムを呼び出してシステムのメモリデータの取得を行う．エンクレイヴと SMM プログラム間でメモリデータの暗号化および整合性検査を行うことで，IDS が取得するシステム情報の盗聴と改ざんを防ぐ．しかし，エンクレイヴが SMM プログラムを呼び出す際には SGX の OCALL を実行してから SMI を発生させる必要があるため，呼び出しのオーバヘッドが大きい．uBPF-SV では，uBPF プログラムを用いることで呼び出し回数を削減している．

eBPFmonitor [17] は，uBPF の基になった eBPF を用いて機密 VM 内の OS データを一括取得することを可能にする．監視対象システムをコンテナに隔離するのは SEV-monitor と同じであるが，エージェントはその外側でプロセスとして実行する．IDS はエージェント経由で eBPF プログラムをカーネルにロードし，eBPF プログラムが OS のデータ構造を解析して監視に必要なメモリデータを収集する．これにより，IDS とエージェント間の通信回数を削減し，監視の高速化を実現している．しかし，カーネル内だけでエージェントを実行する場合よりもさらに安全性は低下する．

00SEVen [18] は，AMD SEV-SNP で導入された VM 特権レベル（VMPL）を利用することで，機密 VM 内で安全にエージェントを実行して監視を行うことを可能にする．エージェントを最も高い特権レベルの VMPL0 で実行し，より低い特権レベルで動作する監視対象システムから保護する．エージェントは VSOCK とプロキシ経由でリモートホストと通信し，IDS から要求されたメモリデータを転送する．uBPF-SV は uBPF プログラムと共有メモリを用いてメモリデータを効率よく収集することができ，SEV-SNP を用いない機密 VM にも適用可能である．

8. まとめ

本稿では，SMI インジェクションと uBPF を用いて機密 VM からメモリデータを取得して監視を行う uBPF-SV を提案した．エージェントを監視対象 VM の BIOS 内に配置し，SMM で動作させることにより，VM 内の侵入者からの攻撃を防ぐ．監視対象システムをエージェントから隔離するためのオーバヘッドが生じないため，エージェントの実行中以外はシステム性能が低下しない．また，BIOS に送り込む uBPF プログラムはロード時に検証を行うことにより安全に実行することができ，IDS が必要とする OS データを効率よく一括取得することができる．実験により，OS データを一括取得することによって一つずつ取得するよりも監視性能を向上させられることが分かった．

今後の課題は，共有メモリ上の複数の結果領域が離れている場合にも未使用領域を除外できるように暗号化し，監視結果の暗号化オーバヘッドを削減することである．また，プロセスリスト以外の分割取得時にも一貫性確認を行えるようにする必要がある．

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] AMD Developer Central: AMD Secure Encrypted Virtualization (SEV), AMD (online), available from ⟨https://www.amd.com/en/developer/sev.html⟩ (accessed 2024-08-03).

[2] Intel Corporation: Intel Trust Domain Extensions (Intel TDX) Introduction and Architecture, Intel Corporation (online), available from ⟨https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/overview.html⟩ (accessed 2024-06-10).

[3] Tal Garfinkel and Mendel Rosenblum: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proc. Network and Distributed System Security Symposium (NDSS)* (2003).

[4] T. Garfinkel and M. Rosenblum: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *In Proceedings of Network and Distributed Systems Security Symposium*, pp. 191–206 (2003).

[5] Tomoharu Nono and Kenichi Kourai: Secure Monitoring of Confidential VMs with Isolated Agents, *Proceedings of the 18th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2025)* (2025).

[6] TianoCore Project: Open Virtual Machine Firmware (OVMF), TianoCore Project (online), available from ⟨https://github.com/tianocore/tianocore.github.io/wiki/OVMF⟩ (accessed 2024-06-10).

[7] The QEMU Project Developers: QEMU documentation, QEMU Project (online), available from ⟨https://www.qemu.org/docs/master/⟩ (accessed 2024-06-10).

[8] IO Visor Project: Userspace eBPF VM, IO Visor Project (online), available from ⟨https://github.com/iovisor/ubpf⟩ (accessed 2024-

ⓒ2026 Information Processing Society of Japan

07-22).

[9] Kazuki Takiguchi and Kenichi Kourai: Nested SEV: Secure and Generic SEV Support for Nested Virtualization, *In Proceedings of the 20th USENIX Symposium on Operating Systems Design and Implementation (OSDI'26)*, pp. 1051–1066 (2026).

[10] The Linux Kernel: BPF Documentation, The Linux Kernel (online), available from ⟨https://docs.kernel.org/bpf/⟩ (accessed 2024-07-22).

[11] Yuichi Ozaki, Sousuke Kanamoto, Hiroaki Yamamoto, and Kenichi Kourai: Detecting System Failures with GPUs and LLVM, *In Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys 2019)*, pp. 47–53 (2019).

[12] Joanna Rutkowska and Rafał Wojtczuk: Preventing and Detecting Xen Hypervisor Subversions, *Blackhat Briefings USA* (2008).

[13] Fengwei Zhang, Jiang Wang, Kun Sun and Angelos Stavrou: HyperCheck: A Hardware-Assisted Integrity Monitor, *IEEE Transactions on Dependable and Secure Computing*, Vol. 11 (2014).

[14] Ahmed M. Azab, Peng Ning, Zhi Wang, Xuxian Jiang, Xiaolan Zhang and Nathan C. Skalsky : HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity, *In Proceedings of the 17th ACM conference on Computer and communications security*, pp. 38–49 (2010).

[15] Yoshimichi Koga and Kenichi Kourai: SSdetector: Secure and Manageable Host-based IDS with SGX and SMM, *In Proceedings of the 22nd IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom-2023)*, pp. 539–548 (2023).

[16] Intel Corporation: Intel Software Guard Extensions (SGX), Intel Corporation (online), available from ⟨https://software.intel.com/en-us/sgx⟩ (accessed 2024-09-09).

[17] Kanta Uesugi and Kenichi Kourai: Secure and Efficient Monitoring of Confidential VMs Using Programmable Read-ahead with eBPF, *In Proceedings of the 19th IEEE International Conference on Cloud Computing (CLOUD 2026)*, pp. 493–499 (2026).

[18] Fabian Schwarz and Christian Rossow: 00SEVen - Reenabling Virtual Machine Forensics: Introspecting Confidential VMs Using Privileged in-VM Agents, *Proceedings of the 33rd USENIX Security Symposium* (2024).

ⓒ2026 Information Processing Society of Japan