SMIインジェクションとuBPFを用いた機密VMの高速な監視

山口 紘輝[1]　末永 道正[1]　光来　健一[1]

概要：クラウドによって提供される仮想マシン (VM) の利用が増加するにつれて，クラウドの内部犯による VM 内の機密情報の盗聴や改ざんが問題になっている．この問題に対して，最近のクラウドは CPU の隔離実行環境（TEE）を用いて保護された機密 VM を提供している．機密 VM であっても侵入検知システム（IDS）を用いて監視を行う必要があるが，VM 外で VM のメモリから安全に OS データを取得する IDS は利用することができない．先行研究では，VM 内で安全に動作するエージェント経由でメモリデータを取得することにより監視を可能にしている．しかし，エージェントの配置の方法によって，安全性やシステム性能にトレードオフが生じる．本稿では，このトレードオフを解消するために，エージェントを BIOS 内でシステムマネジメントモード（SMM）を用いて動作させ，uBPF を使用して OS データの取得を行う uBPF-SV を提案する uBPF-SV は VM 外からシステムマネジメント割り込み（SMI）を挿入する SMI インジェクションを用いてエージェントを呼び出す．さらに，BIOS に uBPF プログラムを送り込んで安全に実行することで，監視に必要なメモリデータを一括取得する．OS データを一つずつ取得する手法との比較を行った結果，uBPF-SV は OS データを大幅に高速に取得できることが分かった．

1. はじめに

近年，クラウドにおいては仮想マシン（VM）を用いるのが一般的になっており，その中で機密情報を扱う機会も増加している．一方で，クラウドへの侵入者やクラウドの内部犯による VM 内の機密情報の盗聴や改ざんといったセキュリティリスクが問題視されている．実際，情報処理推進機構（IPA）の情報セキュリティ 10 大脅威 2025 [1] では，内部不正による情報漏洩等が第 4 位に挙げられている．これを受けて，最近のクラウドでは，CPU が提供する隔離実行環境（TEE）を用いて保護された機密 VM が提供されるようになっている．機密 VM のメモリや CPU レジスタは VM 外からアクセスすることができないため，VM 内の機密情報を守ることができる．

機密 VM は外部からの盗聴や改ざんは防げるものの，VM 内部ではメモリや CPU レジスタが復号されるため，VM 内に侵入された場合には通常の VM と同じリスクにさらされる．そのため，VM への侵入を検知する侵入検知システム（IDS）が必要となる．IDS を VM 内部で実行すると侵入者によって無効化される恐れがあるため，VM 外で IDS を動作させる IDS オフロード [2] が用いられている．しかし，機密 VM の場合には VM 外から VM のメモリを解析して OS データを取得することができない．この問題を解決するために，SEVmonitor [3] は機密 VM 内でエージェントを動作させ，IDS がメモリデータを取得できるようにしている．SEVmonitor ではエージェントを VM 内の侵入者から保護する必要があるが，エージェントの配置の方法によって安全性やシステム性能にトレードオフが生じる．

本稿では，エージェントを機密 VM の BIOS 内に配置し，uBPF を用いて OS データを取得する **uBPF-SV** を提案する．エージェントは x86 プロセッサの BIOS のみが使用可能な動作モードであるシステムマネジメントモード（SMM）を用いて安全に動作させる．この **SMM エージェント**が使用するメモリには SMM 以外の動作モードではアクセスできないため，VM 内の侵入者がエージェントを攻撃することはできない．また，SMM エージェントを実行している時以外はシステム性能がまったく低下しない．SMM エージェントは **SMI インジェクション**と呼ぶ機構を用いて VM 外から呼び出す．この機構は VM 外部から VM の仮想 CPU にシステムマネジメント割り込み（SMI）を挿入することを可能にする．

SMI インジェクションのオーバヘッドを減らすために，uBPF-SV は uBPF を用いて 1 回の SMI インジェクションで監視に必要な OS データを一括取得する．そのため

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

に，BIOS に uBPF プログラムを送り込んで実行し，VM のメモリ上の OS のデータ構造を解析して OS データを収集する．収集した OS データは共有メモリを用いて SMM エージェントから IDS に送られる．uBPF プログラムはロード時に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．ただし，システムメモリや共有メモリには直接アクセスすることができないため，ヘルパー関数を用意して安全にアクセスする．uBPF-SV を OVMF [4]，QEMU-KVM [5]，uBPF ランタイム [6] に実装した．実験の結果，複数の OS データを一括取得することにより，監視性能を大幅に向上させられることを確認した．

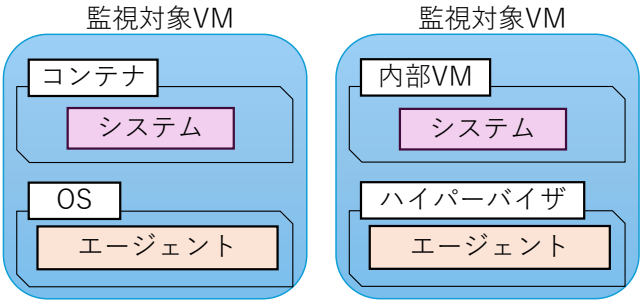
以下, 2 章では機密 VM 内のシステムを監視する際の問題点について述べる．3 章では機密 VM の BIOS 内で SMM を用いてエージェントを安全に動作させ，uBPF を用いて OS データを一括取得する uBPF-SV を提案する．4 章では uBPF-SV の実装について説明し，5 章では uBPF-SV の動作確認と性能測定のために行った実験について述べる．6 章で関連研究に触れ，7 章で本稿をまとめる．

2. 背景

Amazon Web Services，Google Cloud，Microsoft Azure などの最近のクラウドでは，機密 VM と呼ばれる VM が提供されるようになっている．機密 VM は CPU によって提供される AMD SEV [7] や Intel TDX [8] などの隔離実行環境（TEE）を用いて保護された VM である．例えば，SEV は VM のメモリや CPU レジスタを暗号化したり，VM 外から書き換えが行えないようにしたりすることによって VM を保護する．機密 VM を用いることにより，クラウドへの侵入者やクラウドの内部犯から VM 内の機密情報の盗聴や改ざんを防ぐことができる．

機密 VM を利用する場合でも，IDS を用いて VM 内のシステムを監視する必要がある．機密 VM の内部ではメモリや CPU レジスタは復号されており，アクセス制限も行われないため，機密情報の盗聴や改ざんを防ぐことができない．IDS はシステム情報を取得して攻撃者の侵入を検知するが，侵入者によって無効化されると監視を行えなくなる恐れがある．そこで，IDS を VM の外で動作させることにより，VM 内の侵入者から保護する IDS オフロードが用いられている．IDS オフロードは VM イントロスペクションを用いて VM 外から VM のメモリを解析し，OS データの監視を行うことを可能にする．しかし，機密 VM のメモリは暗号化されているため，VM の外からはアクセスすることができない．また，機密 VM の外にオフロードした IDS はクラウドからの攻撃を受ける可能性がある．

そこで，機密 VM に対して IDS オフロードを可能にする SEVmonitor [3] が提案されている．SEVmonitor は VM 内でエージェントを動作させ，エージェント経由で監視に必要なメモリデータを取得する．オフロードした IDS は取得したメモリデータを解析することで OS データの監視を行う．また，IDS を別の機密 VM で動作させることでクラウドから保護する．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通信を行い，機密 VM 内でデータを暗号化することで通信を保護する．SEVmonitor は機密 VM 内で監視対象システムを隔離し，エージェントをその外側に配置することによって侵入者から保護する．しかし，エージェントの配置によって安全性やシステム性能にトレードオフが生じる．



(a) コンテナによる隔離　(b) 内部 VM による隔離

図 1: 2 種類のエージェントの配置

SEVmonitor では，図 1(a) のように，監視対象システムを機密 VM 内のコンテナに隔離し，カーネル内にエージェントを配置することができる．コンテナのオーバヘッドは小さいため，隔離によるシステム性能の低下を抑えることができる．一方，コンテナの隔離は VM よりも弱いため，コンテナ経由でカーネルが攻撃されるとエージェントが無効化される危険性がある．また，安全性のためにコンテナ内のシステムは管理者権限が必要な機能は利用できない．コンテナに管理者権限を与えると，カーネル内のエージェントが攻撃されるリスクが高まる．

図 1(b) のように，監視対象システムを機密 VM の中に作成した内部 VM に隔離し，ハイパーバイザ内にエージェントを配置することもできる．VM の隔離はコンテナよりも強いため，エージェントをより安全に実行することができる．また，内部 VM の中では管理者権限が必要な機能も利用することができる．しかし，ネストした仮想化 [9] を用いて VM 内で VM を動作させるため，二重の仮想化のオーバヘッドによりシステム性能が大幅に低下する．

3. uBPF-SV

本稿では，機密 VM の BIOS 内でシステムマネジメントモード（SMM） [10] を用いてエージェントを安全に動作させ，uBPF を用いて OS データを取得する uBPF-SV を提案する．SMM は BIOS のみが使用可能な 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）を用いて独立した実行環境を提供する．VM の仮想 CPU でも SMM のエミュレーションが行われてお

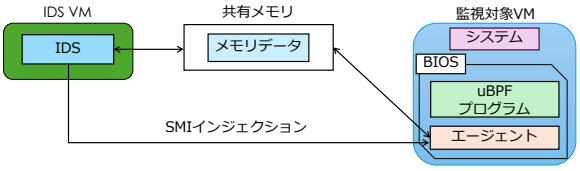


図 2: uBPF-SV のシステム構成

り，SMM 以外の動作モードで動作する監視対象システムは SMRAM にアクセスすることができない．そのため，たとえ機密 VM 内に侵入されたとしても，SMRAM 上に格納されたコードやデータを用いる SMM エージェントを攻撃者が無効化することはできない．また，監視対象システムを SMM エージェントから隔離するためのオーバヘッドは SMRAM のアクセス制御の切り替えのみである．そのため，SMM エージェントを実行して監視を行っている時以外はシステム性能がまったく低下しない．

uBPF-SV では，監視対象 VM の外部で動作する IDS が BIOS 内の SMM エージェントを呼び出す必要があるが，BIOS は通信機能を持たないため，直接的に通信を行うことはできない．IDS が SMM エージェントを呼び出すには，機密 VM 内のプロキシを用いる方法と共有メモリを用いる方法の 2 つが考えられる．前者の方法では，IDS が VM 内で動作するプロキシと通信して要求を送り，プロキシが仮想 CPU に対してシステムマネジメント割り込み（SMI）を発生させることで，SMM エージェントを呼び出すことができる．ただし，プロキシが VM 内部で動作するため，VM 内に侵入された場合には攻撃者によって改ざんされたり無効化されたりする危険性がある．後者の方法では，IDS が監視対象 VM との間に確立した共有メモリに要求を書き込み，SMM エージェントが要求を読み取って処理を行うことができる．しかし，SMM でプログラムを実行する際にはシステム全体を停止させる必要があるため，SMM エージェントを実行し続けて，共有メモリのポーリングを行うことはできない．

そこで，uBPF-SV は SMI インジェクションと呼ぶ機構を用いて SMM エージェントを呼び出す．この機構は VM 外部から VM の仮想 CPU に SMI を挿入することを可能にする．IDS がハイパーバイザに対して SMI の挿入を要求すると，ハイパーバイザが監視対象 VM の仮想 CPU に対して SMI を発生させる．この機構により，VM の BIOS 内に用意された SMI ハンドラが実行される．ただし，SMI インジェクションで発生させる SMI には一般的な SMI のように付加的な情報を渡すことができないため，エージェントを呼び出すための SMI と他の SMI を区別するのが難しい．そこで，IDS が共有メモリに要求を書き込んでいる場合にのみ，SMI ハンドラは SMM エージェントを実行する．

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のはオーバヘッドが大きいため，uBPF を用いて VM のメモリ上の OS のデータ構造を解析することで必要なデータを一括取得する．OS データの解析を柔軟に行えるようにするために，uBPF-SV は監視対象 VM の BIOS 内に uBPF プログラムを送り込んで実行する．uBPF は Linux カーネルで用いられている eBPF[11] のユーザ空間実装であり，バイトコードを安全に実行することができる．そのために，ロード時に uBPF プログラムの検証を行うことで，任意のメモリ領域への不正アクセスなどの危険な動作を防止する．この機構により，IDS が実行時に送り込んだプログラムを BIOS 内で安全に実行することができる．uBPF のバイトコードは JIT コンパイルを用いてネイティブコードに変換することで，高速な実行が可能である．

OS データが格納されたシステムメモリや IDS と SMM エージェント間の共有メモリに uBPF プログラムがアクセスできるようにするために，uBPF-SV はヘルパー関数を提供する．これにより，これらの外部メモリへのアクセスが uBPF プログラムの検証時に行われるメモリ境界チェックによって不正なアクセスとみなされないようにする．uBPF プログラムがヘルパー関数を実行すると uBPF ランタイムが呼び出され，uBPF ランタイム内で外部メモリへのアクセスが行われる．このアクセスを安全に行うために，ヘルパー関数はシステムメモリ上の OS データを読み込むことと，共有メモリにデータを書き込むことのみを許可する．システムメモリには仮想アドレスを物理アドレスに変換してからアクセスするため，アドレス変換に失敗する不正な仮想アドレスへのアクセスを検出することができる．また，共有メモリには先頭からのオフセットを指定してアクセスするため，共有メモリの範囲外へのアクセスも検出することができる．

4. 実装

uBPF-SV を UEFI BIOS のオープンソース実装である OVMF [4]，VM のデバイスエミュレータの QEMU-KVM [5]，uBPF ランタイム [6] に実装した．

4.1 SMI インジェクション

SMI インジェクションを行うために，IDS はまず，図 3 のように，SMM エージェントの呼び出し要求を監視対象 VM との間に確立した共有メモリに書き込む．そして，監視対象 VM のデバイスエミュレータである QEMU と TCP/IP 通信を行い，新たに追加した inject-smi コマンドを送信する．このコマンドの要求は JSON 形式で作成され，QEMU Machine Protocol（QMP） [12] を用いて送信する．QEMU がこのコマンドを受け取ると，KVM のハイパーバイザに SMI の挿入要求を送る．KVM は指定された仮想 CPU に SMI を挿入し，BIOS 内の SMI ハンドラを実行させる．

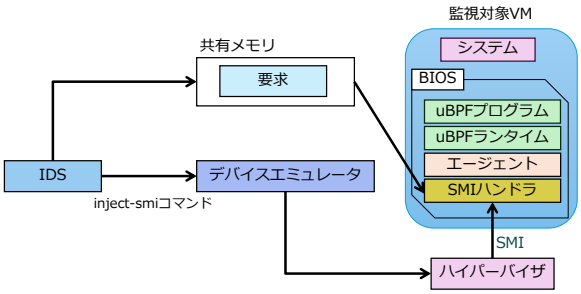


図 3: SMI インジェクション

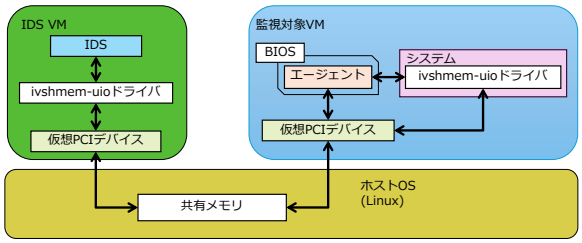


図 4: 共有メモリの構成

4.2 SMM エージェント

BIOS 内の SMI ハンドラは IDS との間の共有メモリを参照し，IDS によって要求が書き込まれていれば SMM エージェントを呼び出す．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，クラウドと VM への侵入者は結託しないことを仮定している．

OVMF は 64 ビットモードで動作するため，SMM エージェントは 4GB を超えるメモリにアクセスすることができる．ただし，セキュリティを高めるためにデフォルトでは SMM でアクセスできるメモリ領域は制限されている．uBPF-SV では，SMM エージェントは攻撃を受けないと仮定しているため，すべてのメモリ領域にアクセスできるようにページテーブルを作成することにより，このメモリアクセス制限を解除する．

4.3 共有メモリを用いた高速通信

IDS と SMM エージェント間で高速に通信を行えるようにするために，IDS VM と監視対象 VM 間に確立した共有メモリを利用する．共有メモリは QEMU が提供する Inter-VM Shared Memory（ivshmem）[13] を用いて作成する．ivshmem は図 4 のようにホスト OS 上のメモリ領域を VM のメモリにマッピングし，共有メモリ用の仮想 PCI デバイスを提供する．VM 内では ivshmem-uio ドライバ [13] をゲスト OS にロードし，共有メモリにアクセスする．IDS VM では，仮想 PCI デバイスを UIO デバイスとして提供して共有メモリをユーザ空間にマップし，IDS プロセスからアクセスできるようにする．IDS は共有メモリに要求を書き込んだ後，SMI インジェクションで要求を SMM エージェントに通知する．SMM エージェントが共有メモリに書き込んだ応答はポーリングで待つことにより取得する．

一方，監視対象 VM では，OS の起動時に ivshmem-uio ドライバが仮想 PCI デバイスから取得した共有メモリの物理アドレスを SMM エージェントに登録する．そのために，ドライバは共有メモリのアドレスをレジスタに格納し，特定の I/O ポートに書き込みを行うことによって SMI を発生させる．uBPF-SV では，ゲスト OS の起動時にはまだ VM に侵入されていないことを仮定しているため，SMM エージェントは登録されたアドレスを信頼することができる．ゲスト OS を完全に信頼しないようにするには，ivshmem-uio ドライバを BIOS 内に実装する必要がある．

SMM エージェントが取得したメモリデータは共有メモリに格納して IDS に返される．その際に，監視対象 VM と IDS VM の外側での盗聴や改ざんを防ぐことができるようにするために，メモリデータは暗号化してから共有メモリに書き込む．そして，IDS が共有メモリ上のメモリデータを復号して OS データを取り出す．

4.4 uBPF ランタイム

SMM エージェントが uBPF プログラムを実行できるようにするために，uBPF ランタイムを BIOS 内で実行できるように移植した．BIOS 内では標準ライブラリが使用できないため，コンパイルに必要となる最小限のヘッダファイルや関数定義を用意した．例えば，`int8_t` などの標準整数型や `bool` 型を定義したり，定数を追加したりした．また，`htobe16` などのエンディアンを変換するための関数を定義したり，`rand_r` 関数を BIOS 内の関数を用いて再実装

したりした．

uBPF ランタイムへの uBPF プログラムのロードは事前に行われ，IDS が共有メモリにロード要求と uBPF プログラムのバイトコードを書き込み，SMI インジェクションを行う．uBPF プログラムは clang を用いてバイトコードにコンパイルする．BIOS 内の uBPF ランタイムはバイトコードをロード時に JIT コンパイルして，高速に実行できる状態にする．この際に，実行可能なメモリ領域が必要となるため，UEFI の SMM Memory Attribute Protocol を用いて専用のメモリを確保し，生成したコードを配置した後で実行不可属性を除去する．これにより，BIOS 内で生成されたコードを実行することができる．

uBPF プログラムを実行する際には，IDS が共有メモリに実行要求を書き込み，SMI インジェクションを行う．これにより，SMM エージェントがすでにロードされた uBPF プログラムを実行し，OS データの収集を行う．uBPF プログラムがシステムメモリや共有メモリにアクセスする際には uBPF ランタイムによって提供されるヘルパー関数を呼び出す．ヘルパー関数は関数番号と関数ポインタの組で登録されており，uBPF プログラムは関数番号を関数のアドレスとして `call` 命令を実行することにより，呼び出すことができる．

uBPF ランタイムは 2 つのヘルパー関数を提供している．一つは図 5 のように，システムメモリ上のデータを共有メモリにコピーするヘルパー関数である．この関数は，システムメモリの仮想アドレス，共有メモリの先頭からのオフセット，コピーするサイズを引数に取る．収集する OS データを含むメモリページを IDS に返すために用いられる．このヘルパー関数が呼び出されると，uBPF ランタイムはまず，監視対象システムのページテーブルの物理アドレスが格納された仮想 CPU の CR3 レジスタの値を取得する．そして，ページテーブルをたどることでシステムメモリの仮想アドレスを物理アドレスに変換し，メモリデータを取得する．

もう一つは，システムメモリ上のデータを取得するヘルパー関数である．この関数は，システムメモリの仮想アドレスを引数として取り，対応するメモリの 64 ビットのデータを返り値として返す．OS のデータ構造のポインタをたどる場合など，uBPF プログラムの中で OS データの処理を行う場合に用いられる．

5. 実験

uBPF-SV が機密 VM から OS データを取得できることを確認する実験を行い，その取得時間を調べた．実験に用いたホストの構成を表 1 に，IDS VM および監視対象 VM の構成を表 2 に示す．なお，本実験では IDS VM と監視対象 VM は機密 VM として実行していない．

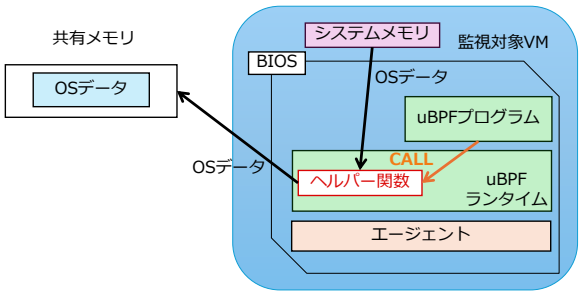


図 5: ヘルパー関数を用いたメモリコピー

表 1: 実験環境（ホスト）

	ホスト
CPU	Intel Core i7-14700
メモリ	192 GB
OS	Ubuntu 22.04.4 LTS
カーネル	Linux 6.8.0-51-generic
仮想化ソフトウェア	QEMU 8.2.7

表 2: 実験環境（VM）

	IDS VM および監視対象 VM
仮想 CPU	1
メモリ	4GB
共有メモリ	1MB
BIOS	EDK II（2024/10/30 版）
OS	Ubuntu 22.04.4 LTS
カーネル	Linux 6.8.0-52-generic
uBPF	2024/11/7 版

```
ids@ids-Standard-PC-Q35-ICH9-2009:~/IDS$ sudo ./IDS3
マップされたアドレス: 0x72ee6e030000
map_region: 0x72ee6e030000
QMPソケットに接続しました
QMPの機能をリクエスト
Result1(linux_banner): Linux version 6.8.0-52-generic (builddd@lcy02-amd64-099) (x86_64-linux-gnu-gcc-12
(Ubuntu 12.3.0-1ubuntu1~22.04) 12.3.0, GNU ld (GNU Binutils for Ubuntu) 2.38) #53~22.04.1-Ubuntu SMP PRE
EMPT_DYNAMIC Wed Jan 15 19:18:46 UTC 2 (Ubuntu 6.8.0-52.53~22.04.1-generic 6.8.12)

Result2(linux_proc_banner): %s version %s (builddd@lcy02-amd64-099) (x86_64-linux-gnu-gcc-12 (Ubuntu 12.3
.0-1ubuntu1~22.04) 12.3.0, GNU ld (GNU Binutils for Ubuntu) 2.38) %s
```

図 6: 2 つの OS バージョン文字列の一括取得

5.1 動作確認

uBPF-SV の動作を確認するために，Linux カーネルのバージョン情報を含む `linux_banner` と `linux_proc_banner` の 2 つの文字列を一括取得する uBPF プログラムを作成した．そして，この uBPF プログラムを SMI インジェクションを用いて BIOS 内の SMM エージェントにロードし，OS バージョン情報の監視を行う IDS を実行した．実行結果は図 6 に示すようになり，1 回の要求で 2 つの文字列を取得できることが確認できた．

次に，システムメモリの不正な仮想アドレスにアクセスする uBPF プログラムを作成し，その uBPF プログラムを用いる IDS を実行した．この uBPF プログラムがシステムメモリのデータを共有メモリにコピーするヘルパー関数

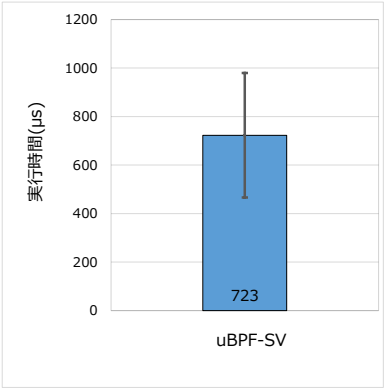


図 7: SMI インジェクションに要する時間

を呼び出した時，uBPF ランタイムはアドレス変換に失敗し，エラーを示す値を共有メモリに書き込んだ．このように，不正なメモリアクセスから BIOS を保護できることが確認できた．

5.2 SMI インジェクションにかかる時間

SMI インジェクションにかかる時間を調べた．IDS が要求を共有メモリに書き込んでから，SMM エージェントが何もせずに完了通知を共有メモリに書き込み，IDS をそれを受信するまでの時間を測定した．実行結果は図 7 に示すようになり，SMI インジェクションには時間がかかり，ばらつきも大きいことが分かった．これは，IDS が QEMU と通信し，QEMU がハイパーバイザを呼び出す処理が行われるためである．

5.3 監視性能

2 つの OS バージョン情報を一括取得するのにかかる時間を調べた．IDS が実行要求を SMI インジェクションによって送信してから，uBPF プログラムが共有メモリに格納したメモリデータを受信して表示するまでの時間を測定した．比較として，uBPF プログラムを用いずに 2 つの OS データを BIOS 内の SMM エージェント経由で 1 つずつ取得する手法についても測定を行った．それぞれの取得時間を図 8 に示す．この結果より，uBPF-SV は一括取得により監視性能を 58%向上させられることがわかった．

JIT コンパイルの効果を調べるために，JIT ありの場合と JIT なしの場合に OS データ取得にかかる時間を測定した．実験の結果は図 9 に示すようになり，JIT ありの場合には実行時間が 8%短縮されることが分かった．この実験では小さな uBPF プログラムを用いたため，性能向上は限定的であったが，uBPF プログラムが複雑になるほど JIT の効果は大きくなると考えられる．

より多くの OS データを一括取得する場合の性能を見積もるために，1 つの OS データ（linux_banner）を 10 回取得する uBPF プログラムを用いた場合の性能を測定した．個別に SMI インジェクションを 10 回行う手法との取得時

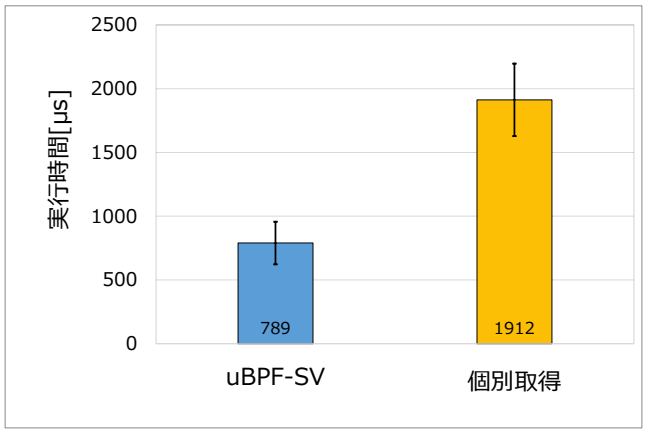


図 8: 2 つの OS データの一括取得性能

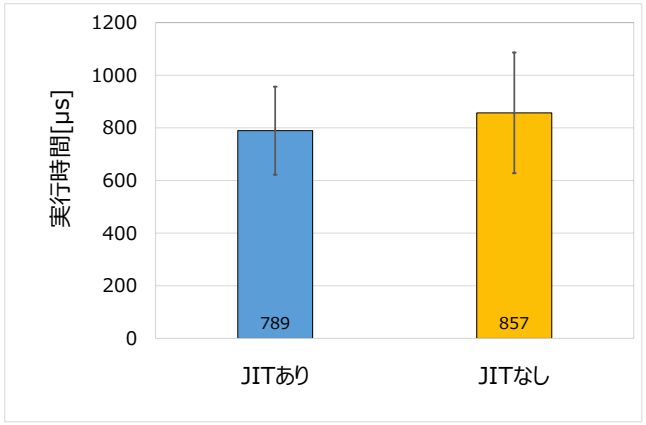


図 9: JIT の有無による実行時間の比較

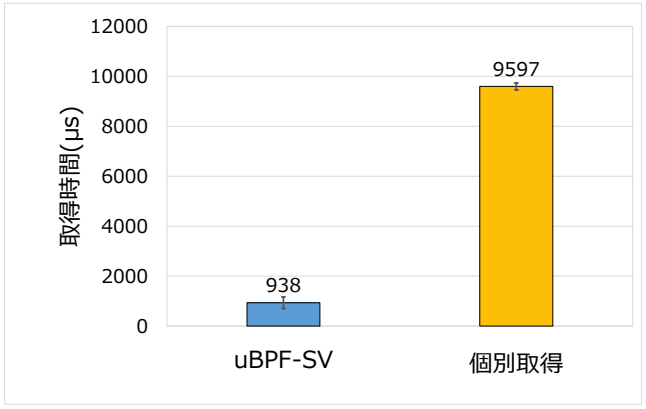


図 10: 10 個の OS データの一括取得性能の見積もり

間の比較を図 10 に示す．uBPF-SV を用いて一括取得を行うことにより，取得時間が平均で 90%削減される可能性があることが確認できた．

5.4 uBPF プログラムのロード時間

uBPF プログラムのロードに要する時間を調べた．IDS が共有メモリに uBPF プログラムを格納し，ロード要求を SMI インジェクションによって送信してから，ロードの完了通知を受信するまでの時間を測定した．JIT コンパイルを行うかどうかで SMM エージェントでのロード時間が変

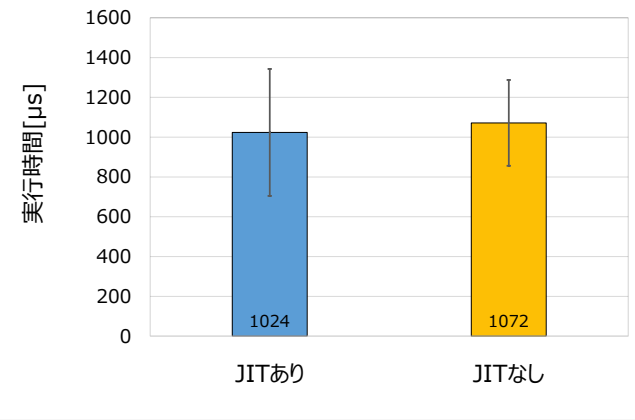


図 11: ロード時間

わるため，JIT ありの場合と JIT なしの場合について調べた．実験結果は図 11 のようになり，ロード処理にかかる時間は JIT ありの場合の方が少し短くなったが，ばらつきが大きいためほとんど差はないと考えられる．

6. 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[14] 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[15] 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 [16] 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 [17] は，SMM と Intel SGX [18] 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[19] 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 [20] 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．

7. 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ないため，エージェントの実行中以外はシステム性能がまったく低下しない．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

取得するよりも大幅に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，OS データの要求や取得に使用している共有メモリの暗号化を行うことで，盗聴や改ざんを防げるようにすることである．また，uBPF プログラムを用いて OS のデータ構造を解析しながらメモリデータを取得できるようにする予定である．

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] 情報処理推進機構: 情報セキュリティ 10 大脅威 2025, 独立行政法人 情報処理推進機構 (online), available from ⟨https://www.ipa.go.jp/security/10threats/10threats2025.html⟩ (accessed 2025-06-12).

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

[3] 能野智玄, 光来健一: 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, **コンピュータセキュリティシンポジウム（*CSS*）** (2022).

[4] 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).

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

[6] IO Visor Project: Userspace eBPF VM, IO Visor Project (online), available from ⟨https://github.com/iovisor/ubpf⟩ (accessed 2024-07-22).

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

[8] 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).

[9] Muli Ben-Yehuda, Michael D. Day, Zvi Dubitzky, Michael Factor. Nadav Har'El, Abel Gordon, Anthony Liguori, Orit Wasserman and Ben-Ami Yassour: The Turtles Project: Design and Implementation of Nested Virtualization, *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 423–436 (2010).

[10] Intel Corporation: Platform Innovation Framework for EFI – System Management Mode Core Interface Specification v0.9, Intel Corporation (online), available from ⟨https://www.intel.com/content/dam/www/public/us/en/documents/reference-guides/efi-smm-cis-v09.pdf⟩ (accessed 2024-09-07).

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

[12] The QEMU Project Developers: QEMU Machine Protocol Specification, QEMU Project (online), available from ⟨https://www.qemu.org/docs/master/interop/qmp-spec.html⟩ (accessed 2024-10-08).

[13] The QEMU Project Developers: Inter-VM Shared Memory device, QEMU Project (online), available from ⟨https://www.qemu.org/docs/master/system/devices/ivshmem.html⟩ (accessed 2024-10-08).

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

[15] 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).

[16] 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).

[17] 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).

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

[19] 上杉貫太, 光来健一: eBPF を用いた Confidential VM の安全かつ高速な監視, *ComSys* (2024).

[20] 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).