RISC-V CoVEにおける TEE VMの柔軟で効率のよい監視システム

梶原 悠大[1]　光来 健一[1]

概要：IaaS 型クラウドにおいて仮想マシン (VM) 内で機密情報を扱うようになっているが，内部犯に機密情報を盗聴されるリスクがある．そこで，最近のクラウドは Trusted Execution Environment (TEE) を利用した Confidential VM (CVM) を提供して VM を保護している．RISC-V でも VM 向けの TEE として CoVE を提供しており，TEE VM (TVM) を安全に実行することができる．しかし，TVM 内に侵入されると CoVE による保護は無力となる．本稿では，CoVE において TVM の柔軟かつ効率のよい監視を実現する TVMmonitor を提案する．TVMmonitor は監視用 TVM を用いて監視システムを実行し，監視対象 TVM から情報を取得することにより侵入を検知する．監視用 TVM と監視対象 TVM が高速に通信を行えるように，TVMmonitor は TVM 間の共有メモリを提供する．CoVE のエミュレーション環境に TVMmonitor を実装し，プロセス一覧が取得できることを確認した．

1. はじめに

IaaS 型クラウドの普及が進む中，利用者はインターネット経由で仮想マシン (VM) を活用して機密情報を含む様々なシステムを構築している．しかし，クラウド内の悪意をもつ管理者などの内部犯による VM 内の機密情報の盗聴リスクが問題となっている．例えば，内部犯にハイパーバイザの脆弱性を悪用されると，VM のメモリやディスクに不正にアクセスされる恐れがある．情報処理推進機構 (IPA) の情報セキュリティ 10 大脅威 2024 [1] では内部不正による情報漏洩が第 3 位に挙げられている．そのため，完全には信用することのできないクラウドから VM を保護する必要性が高まっている．そこで最近は，AMD SEV [2] や Intel TDX [3] のような Trusted Execution Environment (TEE) と呼ばれるハードウェアで保護された Confidential VM (CVM) を提供しているクラウドもある． CVM のメモリはアクセスを制限されたり暗号化されたりしているため，クラウドの内部犯でさえ CVM のメモリ上の機密情報を盗聴することはできない．

命令セットアーキテクチャ (ISA) がオープンソースで提供されている RISC-V は VM 向けの TEE として CVM 拡張 (CoVE) [4] を提供している．CoVE ではメモリが隔離された TEE VM (TVM) を CVM として実行することができ，TEE セキュリティマネージャ (TSM) と呼ばれるソフトウェアが TVM に対するメモリアクセス制御を行う．しかし，CoVE のメモリ隔離は TVM 外部からの不正メモリアクセスのみを防ぐため，攻撃者は TVM に侵入すればメモリ上の機密情報にアクセスすることができてしまう．そのため，TVM への攻撃者の侵入を検知する必要がある．TVM 内で監視を行うと侵入者に無効化される恐れがあるため，TVM の外で攻撃の影響を受けずに監視を行うことが望ましい．SEV については CVM の外から監視を可能にするシステムが提案されている [2] が，CoVE にそのまま適用できるかは不明である．

本稿では，CoVE において TVM の柔軟かつ効率のよい監視を実現するシステム TVMmonitor を提案する．TVMmonitor は監視用 TVM において監視システムを実行し，監視対象 TVM から情報を取得することにより監視を行う．監視用 TVM は TSM によって監視対象 TVM から隔離されるため，監視対象 TVM からの攻撃を防ぐことができる．TSM のような信頼できるシステムソフトウェアがない SEV を用いるシステムと違い，CoVE では TSM の中で TVM を監視することも可能である．しかし，監視用 TVM を用いることでより柔軟に監視を行うことができる．また，TVM 間で共有メモリを用いて情報を取得することにより，ネットワーク通信を用いるより高速かつ安全に監視を行うことができる．

監視用 TVM と監視対象 TVM の間で共有メモリを確立するために，TVMmonitor はそれぞれの TVM に割り当てられた機密メモリの一部を非機密メモリに変換する．監視

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

用 TVM は変換したメモリを共有に用いるメモリとして登録し，監視対象 TVM は自身のメモリをそのメモリで置き換える．許可された TVM 間でのみメモリを共有できるようにするために，メモリの登録・置換の際には認証キーを指定させる．非機密メモリは TVM 以外からもアクセスできるため，TVM 内のソフトウェアが共有メモリの暗号化を行う．TVMmonitor を CoVE のエミュレーション環境 [5] に実装し，プロセス一覧が取得できることを確認した．

以下，2 章では RISC-V の CVM 拡張と TVM の監視の必要性について述べる．3 章では TVM の柔軟かつ効率のよい監視を実現する TVMmonitor を提案する．4 章では TVMmonitor の実装について説明する．5 章では TVMmonitor 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 章で関連する技術や研究に触れ，7 章で本稿をまとめる．

2. RISC-V CoVE における監視の必要性

最近のクラウドはデータの機密性を保つことができる Confidential VM (CVM) を提供している．CVM は AMD SEV や Intel TDX のような VM 向けの TEE で保護されており，クラウドの内部犯による機密データの盗聴を防ぐことができる．TEE は信頼できる環境の中で機密データを用いた処理を行えるようにするためのハードウェアベースの技術である．VM 向けの TEE は VM の中で TEE に対応した OS と既存のアプリケーションの実行を可能にし，VM のメモリ全体を暗号化したりアクセス制御を行ったりすることにより VM を保護する．メモリの暗号化に用いる鍵はハードウェアによって安全に管理される．そのため，VM を動作させているハイパーバイザでさえ VM のメモリ上の機密データにアクセスすることはできない．

RISC-V でも VM 向けの TEE として CVM 拡張 (CoVE) [4], [6] が提供されている．RISC-V はオープンソースで提供されている命令セットアーキテクチャであり，組込み機器からサーバまでの広い範囲を対象としている．**図 1** に CoVE のシステム構成を示す．マシンモード (M-mode) で動作するファームウェアに TEE セキュリティマネージャ (TSM) のドライバが組み込まれる．この TSM ドライバが Memory Tracking Table (MTT) を用いてシステムを TEE と非 TEE に分割する．MTT は TEE からしかアクセスできない機密メモリと，非 TEE からでもアクセスできる非機密メモリを提供することを可能にするハードウェアである．機密メモリを暗号化することにより，メモリに対する物理的な攻撃を防ぐこともできる．

TEE 側では RISC-V のハイパーバイザ拡張を用いて TEE VM (TVM) と呼ばれる CVM を動作させる．ハイパーバイザ拡張スーパバイザモード (HS-mode) で TSM が動作する．TSM は必要最小限の機能のみを提供するソフトウェアであり，TVM のメモリアクセス制御を行う．TVM

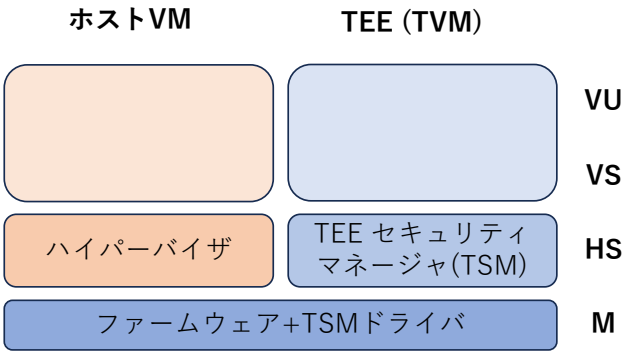

図 1 RISC-V CoVE のシステム構成

内では仮想スーパーバイザモード (VS-mode) で OS が動作し，仮想ユーザモード (VU-mode) でアプリケーションが動作する．TVM 同士は TSM によって管理される G-stage ページテーブルを利用して隔離される．仮想アドレスを物理アドレスに変換するために用いられる TVM 内のページテーブルとは異なり，G-stage ページテーブルは TVM 内のゲスト物理アドレスをホスト物理アドレスに変換するために用いられる．

一方，非 TEE 側はホスト環境であり，HS-mode でハイパーバイザが動作する．このハイパーバイザは通常の VM と TVM のリソースを管理する．その上で動作するホスト VM 内では VS-mode と VU-mode で OS とアプリケーションがそれぞれ動作する．ハイパーバイザとファームウェア内の TSM ドライバの間および，TVM と TSM の間にはアプリケーション・バイナリ・インタフェース (ABI) が定義されている．前者の ABI は CoVE Host ABI (COVH-ABI) と呼ばれ，TVM の作成，TVM へのメモリや仮想 CPU の追加，仮想 CPU のスケジューリングなどを行うためのインタフェースである．後者の ABI は CoVE Guest ABI (COVG-ABI) と呼ばれ，TVM とホスト間でのメモリ共有，TVM のアテステーションや I/O などを行うためのインタフェースである．

このように TVM は機密メモリを用いることによって内部犯から保護され，G-stage ページテーブルを用いることによって他の TVM からも隔離されるが，攻撃者は TVM に侵入することで機密メモリ上の情報にアクセスすることができる．TVM がインターネット経由でサービスを提供している場合，不特定多数の攻撃者からの攻撃にさらされる．マルチテナントのクラウドでは他の TVM からの攻撃を受ける可能性もある．そのため，TVM への攻撃者の侵入を監視する必要があるが，監視対象 TVM の外で攻撃の影響を受けずに監視を行えるようにすることが望ましい．TVM の中で監視を行うと侵入者に無効化されるリスクがあるためである．SEV については CVM の外から監視を行う手法が提案されている [7] が，アーキテクチャの異なる CoVE にそのまま適用できるかは不明である．

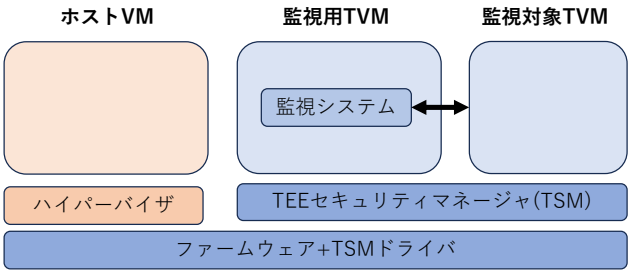

図 2 TVMmonitor のシステム構成

3. TVMmonitor

本稿では，CoVE において TVM の柔軟かつ効率のよい監視を実現する TVMmonitor を提案する．TVMmonitor は監視用 TVM において監視システムを実行し，監視対象 TVM から情報を取得することにより監視を行う．監視用 TVM のメモリは TSM の G-stage ページテーブルを用いて他の TVM から隔離されるため，監視対象 TVM や他のテナントが使っている TVM からの攻撃を防ぐことができる．また，複数の監視用 TVM を用いて監視を行うことも可能であり，分散した監視システムを構築してシステム全体のセキュリティを高めることができる．TSM のような信頼できるシステムソフトウェアが存在しない SEV を用いるシステムと異なり，CoVE では TSM で TVM の監視を行うことも可能である．しかし，TSM の機能は TVM のセキュリティのための必要最小限のものに限定されている．そのため， TSM 内で監視を行うよりも監視用 TVM を用いる方がより高度な監視を行うことができる．

TVMmonitor は監視用 TVM のメモリを共有して通信を行うことにより，監視用 TVM が監視対象 TVM の情報を高速に取得することを可能にする．監視用 TVM と監視対象 TVM が仮想ネットワークを用いて通信を行うことも可能であるが，TVM が仮想 NIC にアクセスする際にはハイパーバイザを経由するため通信のオーバヘッドが大きくなる．また，ハイパーバイザによって通信を妨害される可能性もある．一方で，CoVE は TVM が I/O 等のためにホストと非機密メモリを共有する機構しか提供しておらず，TVM 間で機密メモリを共有することはできない．これはセキュリティのために機密メモリを 1 つの TVM だけに割り当てるようにしているためである．

そこで，TVMmonitor は TVM とホスト間で非機密メモリを共有する機構を利用して TVM 間の共有メモリを実現する．まず，監視用 TVM と監視対象 TVM が TSM を呼び出し，指定したそれぞれのメモリページを機密メモリから非機密メモリに変換する．TSM はハイパーバイザを呼び出すことにより，それらのページをホストと共有する．次に，ハイパーバイザが管理する TVM へのメモリ割り当てにおいて，監視対象 TVM のページを監視用 TVM のページで置き換える．TSM がそれを監視対象 TVM の G-stage ページテーブルに反映することで，監視用 TVM のページが監視対象 TVM と共有される．

特定の監視対象 TVM だけが監視用 TVM のページを共有できるようにするために，TVMmonitor は共通の認証キーが指定された場合にのみページの共有を許可する．監視用 TVM は共有するページを指定する際に TSM に対して認証のためのキーを指定する．監視対象 TVM も置換するページを指定する際に認証キーを指定する．TSM はこれらの認証キーが一致した場合に限り，監視対象 TVM のページを監視用 TVM のページで置き換える．この認証キーは暗号通信などの手段を用いて 2 つの TVM 間で事前に共有しておく．これにより他の TVM が監視用 TVM のページを共有して，監視対象 TVM が共有メモリに格納した情報を盗聴したり改ざんしたりすることを防ぐ．

共有メモリに格納される監視対象 TVM の情報を保護するために，監視対象 TVM 内で共有メモリの内容を暗号化する．TVMmonitor では TVM 間で共有するメモリは非機密メモリとなっており，ハイパーバイザからもアクセスすることができる．CoVE は機密メモリを透過的に暗号化することができるが，非機密メモリについてはホストと共有されるため，暗号化する場合でもホストと同じ暗号鍵を用いる必要がある．そのため，TVM 内のソフトウェアが共有メモリを暗号化・復号化することで，ハイパーバイザへの情報漏洩を防ぐ．さらに，CoVE が非機密メモリを暗号化しない場合でも，メモリに対する物理的な攻撃による情報漏洩を防ぐことができる．暗号化に必要な鍵は TVM 間で事前または鍵共有プロトコルを用いて共有する．

4. 実装

CoVE に対応した RISC-V の実機はまだないため，TVMmonitor の実装には QEMU 上で動作する CoVE のエミュレーション環境 [5] を用いた．

4.1 システム構成

エミュレーション環境のシステム構成は**図 3** のようになっており，**図 2** に示した CoVE のシステム構成とは少し異なる．M-mode ではファームウェアとして標準の OpenSBI [8] が動作し，TSM ドライバは組み込まれない．HS-mode では TSM として Salus [9] が動作するが，非 TEE 側のハイパーバイザは HS-mode では動作しない．ホスト VM は TSM の上で動作し，ホスト VM 内の VS-mode でハイパーバイザとして KVM が動作する．ホスト VM 内でハイパーバイザを実行可能にするために，TSM はネストした仮想化 [10] の処理を行う．ホスト VM では CoVE に対応した Linux が動作し，KVM の機能を提供する．また，VU-mode ではデバイスエミュレータとして KVMTOOL [11] が動作する．TSM 上では TVM も

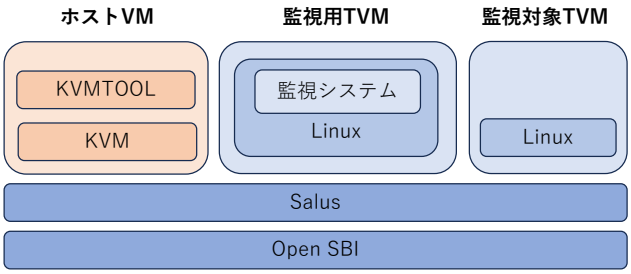

図 3 エミュレーション環境のシステム構成

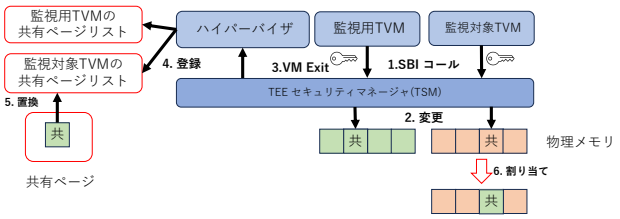

図 4 ページ共有の流れ

動作し，G-stage ページテーブルを用いてホスト VM と TVM および TVM 同士が隔離される．そのため，TEE と非 TEE を隔離するための MTT は使用されていない．

TVMmonitor は監視対象 TVM に加えて監視用 TVM を動作させる．これらの TVM 内では CoVE に対応した Linux が動作する．監視用 TVM 内では Linux カーネルモジュールとして監視システムが動作する．監視対象 TVM 内でも OS 情報を収集してそれを監視システムに提供するために用いられる Linux カーネルモジュールが動作する．これらのカーネルモジュールは TVM 間に確立された共有メモリを用いて通信を行う．

4.2 TVM の作成・実行

CoVE においてメモリはデフォルトでは非機密メモリであり，ハイパーバイザからアクセス可能となっている．そのため，TVM を作成する際にはまず，ハイパーバイザが COVH-ABI を用いて TSM を呼び出し，TVM に必要となるメモリ量を取得する．TSM の呼び出しには Supervisor Binary Interface (SBI) コールが用いられる．次に，取得したサイズの非機密メモリを TSM を呼び出すことで機密メモリに変換する．同時に，その TVM 用の G-stage ページテーブルを作成するために必要なサイズの非機密メモリも機密メモリに変換する．そして，TVM を作成する COVH-ABI を実行すると，TSM は変換した機密メモリを TVM に割り当てる．また，G-stage ページテーブルに使われる機密メモリも同様にしてページテーブル用のページプールに追加される．

その後，ハイパーバイザは TSM を呼び出して TVM の指定したメモリ領域に機密メモリを割り当て，機密メモリにコードやデータを配置して初期化を行う．この際にアテステーションに用いられる計測も行われる．また，ホスト VM との間でメモリを共有するために用いる非機密メモリを TVM の指定したメモリ領域に割り当てる．最後に TVM に仮想 CPU を割り当て，TVM のメモリレイアウトをロックする．それ以降は計測の対象になるメモリは追加することができず，動的に追加することができるのはゼロで初期化されたメモリのみである．そして，ハイパーバイザのスケジューラが TVM の仮想 CPU をスケジューリングした際に，TSM を呼び出して TVM を実行する．

4.3 TVM 間でのページ共有

監視用 TVM と監視対象 TVM 間でページを共有する際にはまず，それぞれの TVM において共有しようとするページを機密メモリから非機密メモリに変換する．そのために，それぞれの TVM は**図 4** のように SBI コールを用いて TSM を呼び出す．TSM は指定されたページの種類を機密から共有に変更し，ホスト VM 内のハイパーバイザに対して VM Exit を発生させる．ハイパーバイザは SBI コールを用いて TSM を呼び出し，指定されたページに対応するエントリを TVM の G-stage ページテーブルから削除する．そして，そのページを TVM の共有ページリストに登録する．この処理には既存の COVG-ABI の sbi_covg_share_memory_region を用いた．

次に，監視用 TVM は共有メモリとして用いるページをハイパーバイザに登録する．監視用 TVM は認証キーを指定して TSM を呼び出し，TSM は認証キーを保存してからハイパーバイザに対して VM Exit を発生させる．ハイパーバイザは指定されたページとそれに対応するゲスト物理アドレスを登録する．この処理のために COVG-ABI を拡張した．その後で，監視用 TVM がそのゲスト物理アドレスにアクセスしようとするとページフォールトが発生し，TSM 経由でハイパーバイザに VM Exit が行われる．ハイパーバイザはページフォールトが発生したゲスト物理アドレスに対応するページを監視用 TVM 用の共有ページリストから探す．そして，TSM を呼び出して TVM の G-stage ページテーブルにそのページに対応するエントリを追加する．このようにして，監視用 TVM は非機密メモリに変換した自身のページにアクセスする．

最後に，監視対象 TVM は自身のページを監視用 TVM のページで置き換える．監視対象 TVM は認証キーを指定して TSM を呼び出し，指定された認証キーが保存しておいたものと一致すればハイパーバイザに対して VM Exit を発生させる．ハイパーバイザは指定されたページのゲスト物理アドレスに対応するページを監視対象 TVM 用の共有ページリストから探す．そして，そのゲスト物理アドレスに対応するページを登録されている監視用 VM のページに変更する．この処理のために COVG-ABI を拡張した．その後で監視対象 TVM がそのゲスト物理アドレスにアク

表 1　マシンの構成

	マシン	エミュレーション環境
CPU	Intel Core i7-12700	RISC-V (2 コア)
メモリ	64 GB	2 GB
OS	Linux 6.5.0	–
QEMU	7.2.50	–
ファームウェア	–	OpenSBI 1.1
TSM	–	Salus

表 2　VM の構成

	ホスト VM	監視用 TVM	監視対象 TVM
仮想 CPU	2	2	2
メモリ	2 GB	320 MB	320 MB
OS		Linux 6.3.0-rc4	
kvmtool	3.18.0	–	–

セスしようとするとページフォールトが発生し，ハイパーバイザが TSM を呼び出して G-stage ページテーブルに監視用 TVM のページに対応するエントリを追加する．このようにして，監視対象 TVM は非機密メモリに変換された監視用 TVM のページにアクセスする．

4.4　監視システム

監視対象 TVM 内で動作しているプロセスの ID と名前の一覧を取得する監視システムをカーネルモジュールを用いて作成した．監視用 TVM と監視対象 TVM にカーネルモジュールがロードされるとまず，TVM 間で共有メモリを確立する．その際に，事前に共有しておいた認証キーを指定する．共有メモリを確立した後，監視用 TVM のカーネルモジュールは共有メモリに情報取得の要求を書き込み，他方の監視対象 TVM のカーネルモジュールは要求が書き込まれるまでポーリングで待つ．その際に CPU 負荷が高くなりすぎないように，200μs 程度のスリープを入れる．監視対象 TVM のカーネルジュールは要求を読み込むとプロセス情報を暗号化して共有メモリに書き込み，その後で応答完了フラグを書き込む．監視用 TVM のカーネルモジュールは応答完了フラグが書き込まれるまでポーリングで待ち，書き込まれたら暗号化されたプロセス情報を読み込んで復号する．暗号化・復号化のための鍵は事前に共有している．

5.　実験

TVMmonior を用いて監視用 TVM だけが監視対象 TVM の OS データを取得できることを確認し，その取得時間を調べる実験を行った．実験に用いたマシンとエミュレーション環境の構成を**表 1** に，VM の構成を**表 2** に示す．

5.1　動作確認

カーネルモジュールとして作成した監視システムを用いて，監視用 TVM から監視対象 TVM のプロセス一覧が

```
/host/root # insmod TVM1.ko
[   43.335174] TVM1: loading out-of-tree module taints kernel.
[   43.335174] TVM1: loading out-of-tree module taints kernel.
[   46.057651] response
[   46.057651] 1: init
[   46.057651] 2: kthreadd
[   46.057651] 3: rcu_gp
[   46.057651] 4: rcu_par_gp
[   46.057651] 5: slub_flushwq
[   46.057651] 6: netns
[   46.057651] 7: kworker/0:0
[   46.057651] 8: kworker/0:0H
[   46.057651] 9: kworker/u4:0
```

図 5　監視用 VM での実行結果 (一部)

共有メモリ経由で取得できることを確認する実験を行った．監視用 TVM と監視対象 TVM にそれぞれのカーネルモジュールをロードし，監視用 TVM が取得したプロセス ID とプロセス名を確認のためにカーネルログに出力した．監視用 TVM における実行結果を**図 5** に示す．この結果より，監視対象 TVM の 48 個のプロセス情報が正しく取得できていることが分かった．また，監視対象 TVM が共有メモリに格納したプロセス情報は暗号化されており，盗聴することはできないことを確認した．

次に，監視用 TVM と監視対象 TVM の間で共有メモリを確立する際に異なる認証キーを指定し，認証キーが一致しない場合にはプロセス情報が取得できないことを確認する実験を行った．この実験の結果，監視対象 TVM が自身のページを監視用 TVM のページで置き換える処理に失敗し，共有メモリの確立を正しく行えないことが分かった．監視対象 TVM が共有メモリにアクセスできないため，監視用 TVM は共有メモリに要求を書き込んだ後，監視対象 TVM からの応答を待ち続けた．

5.2　監視性能

まず，TVMmonitor を用いてメモリを共有するのにかかる時間を測定した．監視用 TVM では，共有メモリに用いるページを確保して非機密メモリに変換し，そのページをハイパーバイザに登録するまでの時間を測定した．監視対象 TVM では，ページを確保して非機密メモリに変換し，そのページを監視用 TVM が登録したページで置き換えるまでの時間を測定した．10 回測定を行った結果を**図 6** に示す．監視用 TVM での処理には監視対象 TVM の 1.9 倍の時間がかかることが分かった．共有メモリの確立に関して，監視用 TVM と監視対象 TVM はほぼ同じ処理を行っているため，処理時間は同程度になると考えられる．このような差が生じた原因は現在，調査中である．

次に，共有メモリを用いて監視用 TVM が監視対象 TVM のプロセス一覧を取得するのにかかる時間を測定した．監視用 TVM がプロセス一覧を要求してから，プロセス情報

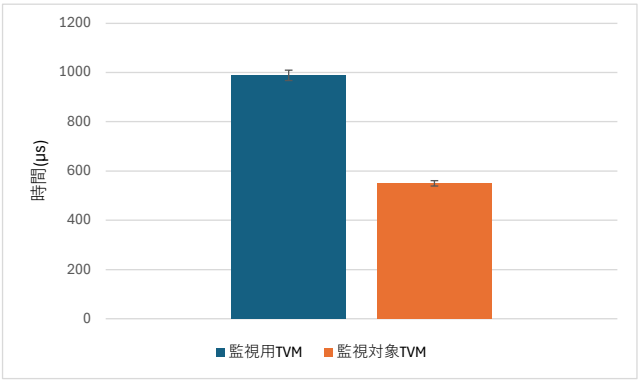

図 6 共有メモリの確立にかかる時間

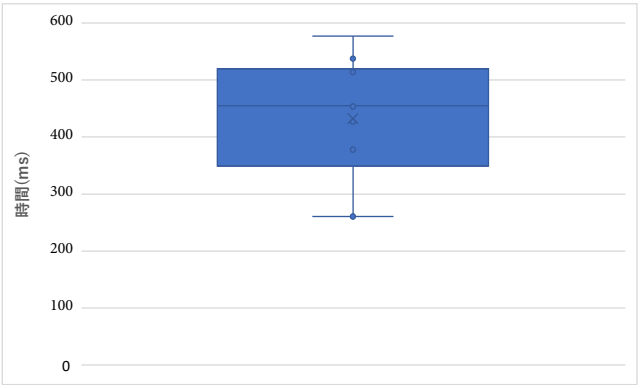

図 7 プロセス一覧の取得時間

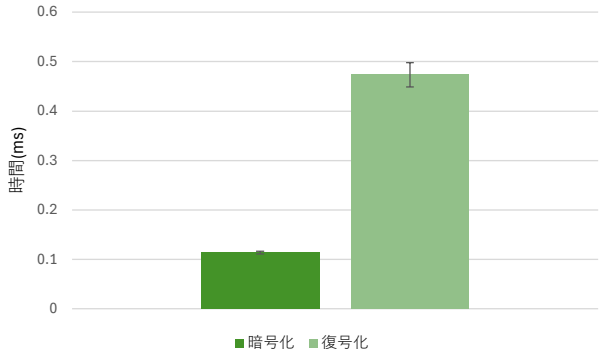

図 8 暗号化・復号化にかかる時間

を取得し終わるまでの時間を 10 回測定した．**図 7** に示すように，48 個のプロセス情報の取得時間は平均 432 ms となり，ばらつきが少し大きいことが分かった．また，監視対象 TVM が共有メモリに格納するデータを暗号化するのにかかる時間および，監視用 TVM が共有メモリのデータを復号するのにかかる時間を測定した．10 回測定を行った結果は**図 8** のようになり，暗号化と復号化を合わせても 587μs であり，プロセス一覧の取得時間全体の 0.1%であった．

6. 関連研究

SEVmonitor [7] は AMD SEV を用いてメモリが暗号化された VM に対して, VM イントロスペクション (VMI) [12] を用いた安全な IDS オフロードを実現している．SEV は VM ごとに固有の暗号鍵を用いることにより，VM 内の機密情報をハイパーバイザや物理的な攻撃から守る．暗号鍵は AMD セキュアプロセッサによって安全に管理される．VM のメモリが暗号化されていると VMI を用いて VM のメモリ上の OS データを取得することができないため，SEVmonitor は監視対象 VM 内でエージェントを動作させる．VM 外にオフロードされた IDS はエージェントと通信を行ってメモリデータを取得し，OS データを解析する．

SEVmonitor は仮想ネットワークまたは共有メモリを用いて IDS と VM 内エージェント間での通信を行う．共有メモリを用いて通信を行う場合は，QEMU の ivshmem 機能 [13] を用いて IDS が動作する VM と監視対象 VM の間で共有メモリを確立する．SEV は VM ごとに異なる暗号鍵を用いるため，ページテーブルエントリの C ビットをクリアすることで共有メモリを暗号化しないようにして共有可能にしている．暗号化していない共有メモリはハイパーバイザからもアクセスすることができるため，TVMmonitor と同様に IDS とエージェントが通信データを暗号化する．

SEVmonitor を用いて VM の監視を行うと通信のオーバヘッドが大きくなるため，通信回数を減らすために eBPFmonitor [14] が提案されている．eBPFmonitor は eBPF プログラムを VM 内の OS に動的に送り込み，OS データを先読みして一括で返送する．eBPF は Linux で性能監視などに使われる仕組みであり，プログラムのロード時に検査器を用いて安全性のチェックが行われるため，OS に影響を与えることなく実行可能である．返送されたデータは IDS がキャッシュに保存し，それ以降はキャッシュにデータがあればエージェントとの通信を行わない．TVMmonitor では監視対象 VM があらかじめ決められたデータを収集して返送しているが，eBPF プログラムを用いて収集することも可能である．

Intel TDX は Secure Arbitration Mode (SEAM) で TDX モジュールを安全に実行し，ハイパーバイザと連携することで CVM を実現する．TDX は VM のメモリを機密メモリと非機密メモリに分割し，それぞれのメモリに対して異なる拡張ページテーブル (EPT) を用いる．機密メモリに対して用いられるセキュア EPT が，CoVE において TSM が管理する G-stage ページテーブルに対応する．SEV とは異なり，ページ単位で暗号鍵を指定することができるため，VM 間で機密メモリを共有して専用の鍵で暗号化するという使い方ができる可能性がある．

Arm CCA [15] は TrustZone が提供するセキュアワールドに加えて，VM を安全に実行することができる Realm ワールドを提供する．Realm ワールドは Granule Protection Table (GPT) を用いてノーマルワールドから隔離されており，ハイパーバイザや通常の VM から保護される．

Realm ワールドでは Realm Management Monitor (RMM) が動作し，ハイパーバイザと通信しながら Realm VM を実行する．CCA と CoVE はよく似ているが，Realm VM の監視についてはまだ研究が行われておらず，Realm VM 間でメモリを共有可能かどうかも不明である．

7. まとめ

本稿では，CoVE において CVM の柔軟かつ効率のよい監視を実現する TVMmonitor を提案した．TVMmonitor は監視用 TVM において監視システムを実行し，共有メモリを用いて監視対象 TVM から情報を取得することにより監視を行う．認証キーが一致する場合のみ TVM 間でのメモリ共有を許可し，TVM 内のソフトウェアが共有メモリを暗号化することによってハイパーバイザによる盗聴を防ぐ．TVMmonitor を CoVE のエミュレーション環境に実装し，監視システムが監視対象 TVM 内のプロセス一覧を取得できることを確認した．

今後の課題は，監視システムをカーネルモジュールではなく OS 上のアプリケーションとして実装できるようにすることである．そのために，監視用 TVM のアプリケーションが監視対象 TVM との間で共有メモリを確立できるようにし，その共有メモリをアプリケーションのメモリにマッピングしてアクセスできるようにする必要がある．また，TVM 間で直接，機密メモリを共有する機構についても検討を行う．CoVE では機密メモリの各ページは 1 つの TVM にしか割り当てられないが，複数の TVM に割り当てる際にはセキュリティへの影響を考慮する必要がある．

謝辞 本研究の一部は，JST，CREST，JPMJCR21M4 の支援を受けたものである．また，本研究成果の一部は，国立研究開発法人情報通信研究機構 (NICT) の委託研究 (JPJ012368C05501) により得られたものである．

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 2024.
[2] Devices, A. M.: Secure Encrypted Virtualization API Version 0.24 (2020).
[3] Intel Corporation: Intel Trust Domain Extensions (2023).
[4] RISC-V International: Confidential VM Extension (CoVE) for Confidential Computing Version 0.6 (2024).
[5] Rivos Inc.: CoVE KVM RISCV64 on QEMU, `https://github.com/rivosinc/cove/`.
[6] Sahita, R., Shanbhogue, V., Bresticker, A., Khare, A., Patra, A., Ortiz, S., Reid, D. and Kanwal, R.: CoVE: Towards Confidential Computing on RISC-V Platforms, *Proceedings of the 20th ACM International Conference on Computing Frontiers* (2023).
[7] 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 2022 (2022).
[8] Rivos Inc.: RISC-V Open Source Supervisor Binary Interface (OpenSBI), `https://github.com/riscv-software-src/opensbi`.
[9] Rivos Inc.: Salus: RISC-V hypervisor for TEE development, `https://github.com/rivosinc/salus`.
[10] Ben-Yehuda, M., Day, M. D., Dubitzky, Z., Factor, M., Har'El, N., Gordon, A., Liguori, A., Wasserman, O. and Yassour, B.-A.: The Turtles Project: Design and Implementation of Nested Virtualization, *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation* (2010).
[11] Deacon, W. and Thierry, J.: Stand-alone Native Linux KVM Tool, `https://github.com/kvmtool/kvmtool`.
[12] Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proceedings of Network and Distributed Systems Security Symposium*, pp. 191–206 (2003).
[13] The QEMU Project Developers: Device Specification for Inter-VM Shared Memory Device, `https://www.qemu.org/docs/master/specs/ivshmem-spec.html`.
[14] 上杉貫太，光来健一：AMD SEV で保護された VM の eBPF を用いた高速な監視，第 159 回 OS 研究会 (2023).
[15] Knight, M. and Stockwell, G.: Arm Confidential Compute Architecture, *Proceedings of the Hardware and Architectural Support for Security and Privacy* (2021).