AMD SEV-SNPで保護された機密VMが制御可能なメモリ監視機構

西村 優志[1]　瀧口 和樹[1]　光来 健一[1]

概要：仮想マシン (VM) を用いてアプリケーションを 1 つだけ実行する Unikernel がクラウド向けに提案されている．Unikernel は VM を隔離実行できる AMD SEV-SNP を用いて保護することにより，機密情報の漏洩を防ぐことができる．このような機密 Unikernel において発生する異常を検知するには Unikernel を監視する必要があるが，機能が最小限に抑えられた Unikernel の内部では高度な監視は難しい．一方，機密 Unikernel のメモリには外部からアクセスできないため，VM イントロスペクションを用いて Unikernel を監視することもできない．本稿では，SEV-SNP によるメモリ保護を Unikernel がきめ細かく制御することにより，Unikernel 外部からの監視を可能にする ShadowMonitor を提案する．ShadowMonitor はユーザによって指定されたポリシに基づいて，Unikernel 内の一部のメモリ領域の暗号化を解除する．それにより，監視機構は VM イントロスペクションを用いて監視に必要なデータを取得することができる．監視機構が Unikernel の仮想アドレスを用いてデータにアクセスできるように，ShadowMonitor はページテーブルを複製して暗号化を解除したシャドウページテーブルを作成する．機密 Unikernel と監視機構を機密 VM の中で実行することで，これらをクラウドから保護することもできる．ShadowMonitor を Nanos と KVMonitor に実装して監視機構の動作を確認し，機密 Unikernel の性能への影響を調べた．

1. はじめに

クラウドにおいては仮想マシン (VM) を用いて単一のサービスのみを実行することが多いため，クラウドでのアプリケーション実行に特化した Unikernel [1] が提案されている．Unikernel は VM 内でライブラリ OS などの軽量 OS を用いてアプリケーションを 1 つだけ実行する．汎用 OS を用いる場合と比べて高速に起動や実行を行うことができるという特長がある．その一方で，クラウドでは内部犯によりサービスの持つ機密情報が盗聴される恐れがある．最近のクラウドでは AMD SEV-SNP [2] や Intel TDX [3] などの Trusted Execution Environment (TEE) を用いて保護された機密 VM が提供されており [4]，メモリを暗号化することによって盗聴を防いでいる．Unikernel も VM であるため，これらの TEE を用いて保護することができる．TEE で保護された Unikernel を本稿では機密 Unikernel と呼ぶ．

機密 Unikernel には通常の Unikernel と同様に様々な異常が発生する可能性があるため，監視を行うことが必要である．TEE は VM 外部からの不正アクセスに対してメモリや CPU レジスタを保護することしかできないため，VM 内に侵入した攻撃者に対しては無力である．また，Unikernel は他のアプリケーションとの間でリソースを融通し合うことができないため，想定以上のリソースを必要とした場合には正常な動作を行うことができない．しかし，Unikernel 内で監視を行うとアプリケーションの異常で監視機構が機能しなくなる可能性があり，Unikernel の軽量さが失われる懸念もある．VM イントロスペクション [5] を用いて Unikernel のデータを VM 外から監視することも考えられるが，メモリが保護されている機密 Unikernel ではこの手法を用いることもできない．

本稿では，SEV-SNP によるメモリ保護を Unikernel がきめ細かく制御し，Unikernel 外部からの監視を可能にする ShadowMonitor を提案する．ShadowMonitor は Unikernel 内の機密情報を含まないメモリ領域の暗号化を解除することで，VM イントロスペクションを用いて監視に必要なデータを取得できるようにする．ユーザは提供されるポリシを選択することにより，暗号化を解除するメモリ領域を容易に指定することができる．ポリシが異なるデータが同じメモリページに配置されると暗号化が解除された際の情報漏洩につながるため，ShadowMonitor はポリシを考慮したメモリ割り当てを提供する．Unikernel はメモリ割り当て時にポリシを指定し，ポリシごとに異なるページを割り当てる．

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

監視機構が Unikernel のデータの仮想アドレスを物理アドレスに変換できるようにするために，ShadowMonitor はページテーブルを複製してシャドウページテーブルを作成する．Unikernel のメモリを解析するには Unikernel のページテーブルを用いてアドレス変換を行う必要があるが，SEV-SNP ではページテーブルが置かれたメモリの暗号化を解除することはできない．そこで，ページテーブルの代わりに，暗号化を解除したシャドウページテーブルを参照することでアドレス変換を可能にする．ページテーブルとシャドウページテーブルを同期するために，ページテーブルの更新時にシャドウページテーブルも更新する．監視機構はハイパーバイザ経由でシャドウページテーブルのアドレスを取得する．

ShadowMonitor は機密 Unikernel と監視機構を機密 VM 内で動かすことで，クラウドから機密 Unikernel のデータと監視機構を保護することもできる．これにより，クラウドは機密 Unikernel が暗号化を解除したデータを改ざんしたり，監視機構を停止したりすることができなくなる．ShadowMonitor を Unikernel の一つである Nanos [6] と VM イントロスペクションのための機構である KVMonitor [7] に実装した．ShadowMonitor を用いて実験を行い，監視機構が機密 Unikernel 内の CPU，メモリ，TCP 通信に関する情報を取得できることを確認した．また，ShadowMonitor のオーバヘッドについても測定した．

以下，2 章では機密 Unikernel を監視する上での問題点を述べる．3 章では機密 Unikernel の監視をメモリ暗号化の制御により実現する ShadowMonitor を提案する．4 章では ShadowMonitor の実装について述べ，5 章では ShadowMonitor を用いて行った実験について述べる．6 章では関連研究について述べ，7 章で本稿をまとめる．

2. 機密 Unikernel の監視

機密 Unikernel は一般的な機密 VM とは異なり，特定のアプリケーションのみを保護することができる．機密 Unikernel に用いることができる TEE の一つである SEV-SNP は，VM のメモリを暗号化することでクラウドの内部犯による盗聴を防ぐ．SEV-SNP は AMD セキュアプロセッサで管理されている暗号鍵を VM ごとに割り当てることにより，それぞれの VM をハイパーバイザから隔離する．SEV-SNP は VM のメモリの暗号化を VM 内のページテーブルで制御しており，ページテーブルエントリ（PTE）の C ビットが 1 の時に対応するページが暗号化され，0 の時には暗号化されない．これは I/O に用いられるバウンスバッファのように VM とハイパーバイザの間でデータの受け渡しを可能にするためである．

SEV-SNP は VM 外部からの不正なメモリアクセスを防ぐことができる一方で，VM 内部でメモリアクセスが行われた時にはデータが常に復号される．そのため，SEV-SNP によるメモリ保護は VM 内に侵入されると無力であり，メモリ上の情報の盗聴を防ぐことはできない．また，少ないリソースしか割り当てないことが多い Unikernel では異常が発生しやすい．例えば，Unikernel に割り当てられたメモリよりも多くのメモリが必要になると，メモリが確保できずに異常終了する可能性がある．複数のアプリケーションが動作する通常の VM ではアプリケーション間でリソースを融通し合うことができるが，単一のアプリケーションしか動作しない Unikernel ではそれも難しい．その結果，リソースを大量に消費させるサービス妨害攻撃にもより脆弱である．

そのため，Unikernel では異常が発生していないかを常に監視して早期に検知することが不可欠である．Unikernel の監視手法として 2 つの手法が考えられる．一つは，Unikernel 内部で監視機構を動作させる手法である．しかし，最小限の機能しか持たない Unikernel 内では柔軟で複雑な監視を行うのは難しいことが多い．Unikernel では軽量 OS と単一のアプリケーションが一体として動作するため，異常が発生すると監視機構が機能しなくなる可能性が高い．特に，多くの Unikernel ではライブラリ OS を用いているため，OS 内で監視機構を動作させてもアプリケーションの異常の影響を受けてしまう．また，Unikernel に監視機能を持たせると，Unikernel の肥大化を招き，軽量さが失われることも懸念される．

もう一つの手法は，Unikernel に対して VM イントロスペクションを用いる方法である．VM イントロスペクションは VM 内のシステムの監視を行うために，VM 外から VM のメモリに直接アクセスしてメモリデータを取得する．そして，VM 内の OS に関する知識を利用してメモリ上の OS データを解析し，監視に必要な情報を取得する．その際に，OS のページテーブルを参照することで OS データの仮想アドレスを物理アドレスに変換してから VM のメモリにアクセスする．この手法は VM を用いて作成されている Unikernel に対しても利用することができる．しかし，機密 Unikernel はメモリが暗号化されているため，監視機構であってもそのメモリにアクセスすることはできない．そのため，機密 Unikernel に対して VM イントロスペクションを用いることはできない．

そこで，機密 VM 内でエージェントを動作させ，VM 外部で監視機構を動作させる SEVmonitor [8] が提案されている．SEVmonitor は監視機構が VM 内のエージェント経由でメモリデータを取得し，メモリデータを解析することで VM 内の情報を取得する．しかし，エージェントと通信を行うオーバヘッドが大きいという問題がある．また，エージェントが監視対象システムの影響を受けないようにする必要があるが，Unikernel ではエージェントを保護するのが難しい．SEVmonitor ではコンテナを用いて監視対象システムを隔離するが，Unikernel はコンテナを提供し

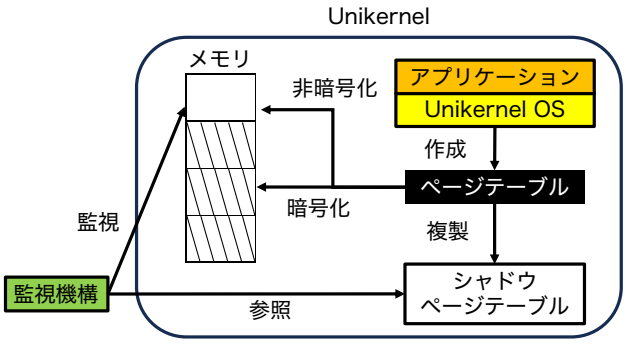

図 1 ShadowMonitor のシステム構成

ない．機密 VM 内部に VM を作成して監視対象システムを隔離することもできるが，Unikernel 内に VM を作成することはできない．

3. ShadowMonitor

本稿では，SEV-SNP によるメモリ保護を Unikernel がきめ細かく制御し，Unikernel 外部からの監視を可能にする ShadowMonitor を提案する．ShadowMonitor のシステム構成は図 1 に示すようになり，監視機構を機密 Unikernel の外部で動作させる．ShadowMonitor は Unikernel の監視に必要であり，かつ，機密情報が含まれないメモリ領域の暗号化を解除する．このようなメモリ領域の例としては，Unikernel のリソース使用状況などのシステム情報やアプリケーションの管理情報，ネットワーク通信に関する情報などが格納されている領域が挙げられる．暗号化を解除するメモリ領域以外は暗号化したままにして保護する．メモリ領域の暗号化の制御は Unikernel のページテーブルを用いてページ単位で行う．

監視機構は暗号化が解除されたメモリ領域に VM イントロスペクションを用いてアクセスして Unikernel のデータを取得し，そのデータを解析することで監視を行う．Unikernel のメモリデータを解析するには，Unikernel のページテーブルを用いて監視対象データの仮想アドレスを物理アドレスに変換する必要がある．しかし，VM 外からメモリ暗号化を解除されないようにするために，SEV-SNP は VM 内のページテーブルが常に暗号化されていることを必要とする．そのため，Unikernel がページテーブルを暗号化しないようにすることはできず，VM 外の監視機構からページテーブルを直接参照できるようにすることはできない．

そこで，ShadowMonitor は Unikernel のページテーブルを複製することにより，シャドウページテーブルを作成する．ShadowMonitor は Unikernel が PTE を作成した時にシャドウページテーブルにも同じ PTE を作成する．PTE を更新した時には同様の更新をシャドウページテーブルに対しても行う．このシャドウページテーブルは MMU によって参照されないため，暗号化しないようにすることができる．VM 外から書き換えることも可能になるが，書き換えられても Unikernel の実行時に行われるアドレス変換には影響を及ぼさず，メモリ領域の暗号化を解除することもできない．

監視機構はシャドウページテーブルを参照することで，機密 Unikernel のデータのアドレス変換を行うことができる．しかし，シャドウページテーブルはページテーブルとは異なり，VM 外からその位置を特定するのが困難である．ページテーブルのアドレスは VM の CPU レジスタから取得できるため，シャドウページテーブルをその隣に配置する方法が考えられる．この方法は SEV では用いることができるが，SEV-SNP では CPU レジスタも暗号化されるため用いることができない．そこで，Unikernel がシャドウページテーブルのアドレスを明示的にハイパーバイザに登録するようにする．監視機構はハイパーバイザからそのアドレスを取得することにより，機密 Unikernel 内のシャドウページテーブルの位置を特定することができる．

暗号化を解除するメモリ領域の指定は Unikernel を実行するユーザが行う．ただし，Unikernel OS 内のデータについてはユーザが詳細に指定するのは難しいため，用意されたいくつかのポリシの中から選択することができる．例えば，メモリ管理に関するデータだけ監視可能にする，ネットワーク通信に関するデータだけ監視可能にする，などである．しかし，SEV-SNP において暗号化の制御はページ単位で行われるため，ポリシが異なる OS データが同じメモリページに配置される可能性がある．そこで，Unikernel OS のメモリ割り当て時にポリシを指定することにより，ポリシごとに異なるページを割り当て，同じページには同じポリシのデータのみが格納されるようにする．

ShadowMonitor は機密 Unikernel と監視機構を機密 VM 内で実行することもできる．機密 VM による保護がない場合，ユーザが動かす監視機構はクラウドによって停止される恐れがある．また，Unikernel 内の暗号化が解除されたデータがクラウドによって改ざんされる恐れもある．機密 VM 内で実行することにより，クラウドからのこのような攻撃を防ぐことができる．機密 VM 内で VM である機密 Unikernel を動かすために Nested SEV [9] を用いる．ただし，ネストした仮想化のオーバヘッドが大きいため，セキュリティとのトレードオフになる．

多くの Unikernel はアプリケーションにライブラリ OS をリンクしており，アプリケーションが攻撃を受けるだけで OS のページテーブルを書き換えることが可能になる．クラウドはアプリケーションの脆弱性を利用してページテーブルを書き換えることさえできれば，メモリの暗号化を解除して機密情報を取得することができる．そこで，ShadowMonitor は汎用 OS と同様にアプリケーションをユーザモードで動作させる Unikernel を用いることで，カーネルモードで動作する OS 内のページテーブルを保護

する．ライブラリ OS とアプリケーションが同一のモードで動く Unikernel を用いる場合には，WebAssembly [10] を用いてアプリケーションを実行することによりライブラリ OS を保護するができる．

4. 実装

ShadowMonitor を Unikernel の一つである Nanos [6] と VM イントロスペクションのための機構である KVMonitor [7] に実装した．Nanos は Linux のバイナリをそのまま実行することができ，アプリケーションをユーザモードで動作させる．

4.1 シャドウページテーブルの作成

ShadowMonitor は Nanos のブートストラップ時にシャドウページテーブルのページディレクトリ用のページを割り当てる．Nanos は Unikernel に SEV-SNP を適用できるようにするために UEFI BIOS を用いて起動し，UEFI のブートサービスを利用して初期メモリを確保する．このメモリから実ページテーブルのページディレクトリ用のページを 1 ページ割り当てるが，その際にシャドウページテーブルのページディレクトリ用のページも 1 ページ割り当てる．

実ページテーブルの更新時にシャドウページテーブルに対しても同じ処理を行うことで実ページテーブルと同期する．ページテーブルは Nanos のカーネル初期化時に様々なメモリ領域が割り当てられる時や，アプリケーション (プロセス) が mmap システムコールで確保したメモリにアクセスして物理メモリが割り当てられる時などに更新される．ページテーブルを更新する際には仮想アドレス，物理アドレス，PTE のフラグが指定される．指定された仮想アドレスを基にページディレクトリからシャドウページテーブルをたどり，更新すべき PTE を見つける．そして，PTE に指定された物理アドレスとフラグをセットする．

PTE の作成時にシャドウページテーブルのページがまだ存在しない時には，実ページテーブルと同様の処理を行って新しいページを割り当てる．ブートストラップ時にはその都度，4KB のページを確保して割り当てる．カーネルの起動時および起動後には性能を向上させるために 2MB の Huge ページを一括で確保し，そこから 4KB ずつページを割り当てる．確保した Huge ページを使い切ったら，再度，Huge ページを確保する．Huge ページの確保は実ページテーブル用とは別に行う．

4.2 シャドウページテーブルの暗号化解除

シャドウページテーブルに新たに 4KB のページを割り当てた時には，監視機構から参照できるようにそのページの暗号化を解除する．まず，ページの物理アドレスに対応する仮想アドレスを取得する必要があるが，取得方法は起動の段階ごとに異なる．ブートストラップ時には仮想アドレスは物理アドレスと同一である．カーネルの起動初期段階ではブートストラップ時に確保した初期メモリが特定の仮想アドレスにストレートマッピングされるため，物理アドレスからベースアドレスを引いたオフセットをその仮想アドレスに足すことで対応する仮想アドレスを計算する．その後は，物理メモリ全体が別の仮想アドレスにストレートマッピングされるため，その仮想アドレスに物理アドレスを足すことで対応する仮想アドレスを計算する．

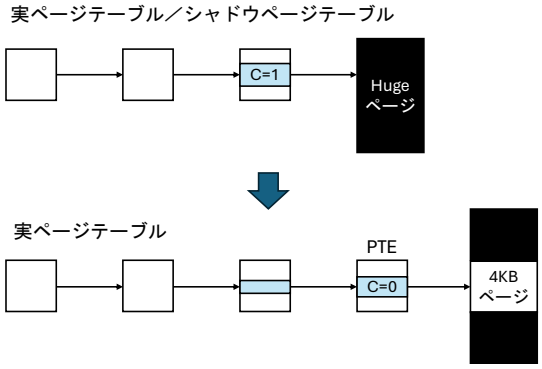

図 2 Huge ページの分割

計算した仮想アドレスを基に実ページテーブルをたどり，対象ページの PTE を見つけてその C ビットをクリアする．SEV-SNP は C ビットが 0 になっている PTE に対応するページは暗号化しない．対象の 4KB のページが 2MB の Huge ページに含まれる場合は，Huge ページに対応する PTE の C ビットをクリアすると Huge ページ全体の暗号化が解除されてしまう．そこで，対象の 4KB のページの暗号化だけを解除するために，図 2 のように Huge ページを 512 個の 4KB のページに分割する．その際に，元の Huge ページの PTE のフラグから NX ビットと C ビットをクリアしたものをレベル 3 のエントリ (PMD) のフラグにし，Huge ページの PTE のフラグから PS ビットをクリアしたものをレベル 4 の PTE のフラグにする．これにより，シャドウページテーブルは実ページテーブルと完全に同一ではなくなるが，アドレス変換は同様に行うことができる．一方，Unikernel 内でのアドレス変換には少し時間がかかるようになる．

上で述べたように，Nanos ではカーネルの起動初期に使われるメモリマッピングとその後で使われるメモリマッピングが異なる．そのため，同じ物理メモリにアクセスするための仮想アドレスが起動途中で変わり，対応する実ページテーブルの PTE も変わる．図 3 のように，変更後のメモリマッピングでは新たに使われる仮想アドレスに対応する PTE の C ビットは 1 に設定されるため，物理メモリ上には暗号化されていないシャドウページテーブルが格納されているにも関わらず，実ページテーブルの PTE の C ビットは 1 という矛盾した状態になる．そこで，シャドウ

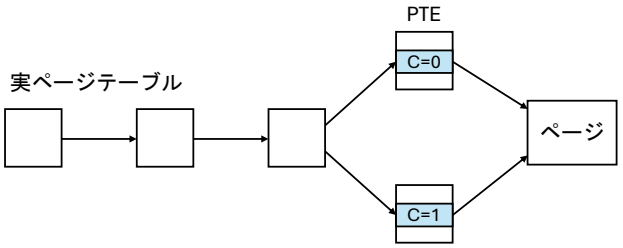

図 3 メモリマッピングの変更

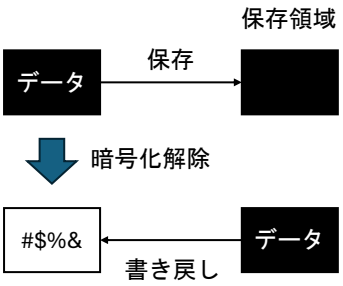

図 4 データが格納されたページの暗号化解除

ページテーブルへのページ割り当て時にその物理アドレスを記録しておく．そして，マッピング変更時にはその物理アドレスに対応する新しい仮想アドレスを取得し，仮想アドレスを基に実ページテーブルをたどって PTE の C ビットを 0 にする．

一方，ブートストラップ時に割り当てられたシャドウページテーブルのページについては，すぐには暗号化を解除することできない．これは PTE を変更することができないように UEFI BIOS によって実ページテーブルが設定されているためである．この設定を変更することはセキュリティの観点から望ましくない．そこで，ブートストラップ時に割り当てられたページについてもその物理アドレスを記録しておく．物理アドレスの記録先をカーネル起動時に容易に特定できるようにするために，初期メモリに確保したシャドウページテーブルのページディレクトリ用のページの次のページに確保する．これにより，カーネルは CR3 レジスタ経由で渡された実ページテーブルのページディレクトリのアドレスから物理アドレスの記録先を特定することができる．

ブートストラップ時に割り当てられたページの暗号化の解除はカーネルの起動後，メモリマッピングが変更される時に行う．しかし，この時にはすでにページにシャドウページテーブルの PTE が暗号化されて書き込まれている．そのため，PTE の C ビットをクリアするだけではページのデータは暗号化されたままとなり，PTE を参照することができなくなる．そこで，図 4 のように暗号化を解除する前にページのデータをコピーして保存しておき，暗号化を解除した後で保存しておいたデータを書き戻す．PTE の C ビットをクリアする前は SEV-SNP によって復号されたデータが読み込まれ，C ビットをクリアした後は SEV-SNP による暗号化を行わずに書き込まれる．

4.3 OS データの暗号化解除

ユーザが指定したポリシによって暗号化が解除される可能性のある OS データについては，Unikernel はそのメモリを割り当てる際に対応するポリシを指定する．指定されたポリシがユーザによって指定されたポリシと一致した場合には，暗号化が解除されたメモリを返す．Nanos は割り当てたメモリをメモリキャッシュを用いて管理しているため，ShadowMonitor ではポリシごとにメモリキャッシュを用意する．指定されたポリシに対応するメモリキャッシュのメモリを返すことで，暗号化されるメモリと暗号化されないメモリを厳密に使い分けることができる．メモリキャッシュからメモリを返せない場合には，ページ単位で新たにメモリを割り当てる．そして，その仮想アドレスを基に実ページテーブルをたどり，PTE の C ビットをクリアすることで暗号化を解除する．

グローバル変数が配置されているメモリ領域については，初期化前に暗号化を解除するのは難しいため，カーネル起動後にページ単位で暗号化を解除する．その際にデータが破壊されないようにするためにデータを保存しておき，暗号化を解除した後で書き戻す．同じページ上のグローバル変数はすべて暗号化されなくなるため，異なるポリシのグローバル変数は別のページに配置されるようにコンパイルを行う必要がある．

現在の実装では，ユーザが指定したポリシに応じて，プロセス，メモリ，ソケット通信に関する情報の暗号化を解除することが可能である．Nanos はカーネル起動時にカーネルプロセスを作成し，その後でアプリケーションを実行するためのプロセスを作成する．この 2 つのプロセスを管理するためのプロセス構造体のアドレスがグローバル変数に格納されているため，プロセスを作成する際にこのグローバル変数が配置されているページの暗号化を解除する．また，動的に確保されたプロセス構造体のメモリ，プロセス構造体からたどれるカーネルヒープ構造体や ID ヒープ構造体のメモリについても暗号化を解除する．これらの構造体にはメモリに関する情報が格納されている．また，通信中のソケット通信の接続先の情報はグローバル変数に格納されているため，このグローバル変数の暗号化も解除する．

4.4 OS データの監視

監視機構が LLView [11] を用いて VM のメモリ上の OS データを解析できるようにするために，Nanos のカーネルからシンボルと仮想アドレスの対応を取得する．LLView はカーネルのグローバル変数や構造体などを用いて監視機構を開発するためのフレームワークである．LLView は LLVM を用いて監視機構のプログラムをコンパイルして中

間表現を生成し，取得したシンボル表を用いて監視機構が参照しているカーネルのグローバル変数を対応する仮想アドレスに置換する．また，メモリからの読み込み命令の前に仮想アドレスを物理アドレスに変換し，さらにホストアドレスに変換する処理を挿入する．

監視機構は KVMonitor [7] を用いて Unikernel のメモリから指定した仮想アドレスのデータを取得する．従来の KVMonitor は VM を動作させている QEMU に QMP コマンドを送信して仮想 CPU の CR3 レジスタの値を取得し，実ページテーブルの物理アドレスを取得していた．しかし，SEV-SNP で保護された機密 VM の場合は CR3 レジスタの値は暗号化されているため，監視機構はアドレスを取得することができない．そのため，Unikernel は起動時にハイパーコールを用いてシャドウページテーブルの物理アドレスをハイパーバイザに登録する．ハイパーバイザがハイパーコールの引数を受け取れるようにするために，SEV-SNP によって暗号化される CPU レジスタではなく，Guest Host Communication Block (GHCB) と呼ばれる暗号化が解除されたメモリ領域に格納して VMGEXIT 命令を実行する．KVMonitor はホスト OS が提供するデバイスファイルを読み込むことで，ハイパーバイザに登録されたシャドウページテーブルの物理アドレスを取得する．

KVMonitor は実ページテーブルを用いる場合と同様に，シャドウページテーブルを参照して OS データの仮想アドレスを物理アドレスに変換する．さらに，得られた物理アドレスをホストアドレスに変換する．KVMonitor は VM のメモリをメモリファイルとして作成するため，Unikernel のメモリファイルをプロセスのメモリ上にマッピングし，ホストアドレスが指すメモリにアクセスする．ただし，機密 VM 内で機密 Unikernel を動作させる場合，Nested SEV では特殊なデバイスファイルを VM のメモリとして使用しており，このデバイスファイルはプロセスのメモリ上にマッピングすることができない．そこで，メモリファイルの代わりに，プロセスのメモリにアクセスするために Linux が提供している`/proc/PID/mem`（`PID` はプロセスの ID）を用いる．機密 Unikernel に対応する QEMU のプロセスのメモリをマッピングすることにより，機密 Unikernel のメモリにアクセスする．

5. 実験

ShadowMonitor を用いて，機密 VM 内で動作する監視機構が機密 Unikernel 内のデータを取得できることを確認し，機密 Unikernel の起動時間や実行時間への影響を調べた．実験には，AMD EPYC 7443P の CPU，128GiB のメモリ，2TB の SATA HDD を搭載したマシンを用いた．ホスト OS として Linux 6.1.0-rc4-snp，仮想化ソフトウェアとして QEMU-KVM 7.1.50 を動作させた．機密 VM には 4 個の仮想 CPU と 2GiB のメモリを割り当て，ゲスト

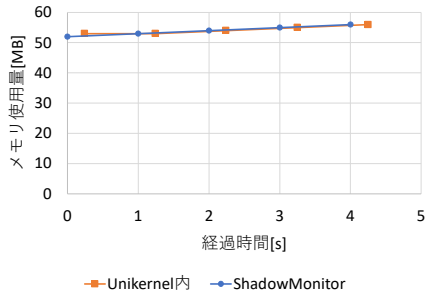

図 5 Unikernel 内外で取得したメモリ使用量

OS として Linux 6.1.0-rc4-snp，仮想化ソフトウェアとして QEMU-KVM 7.1.50 を動作させた．機密 Unikernel として Nanos 0.1.46 を動作させ，1 個の仮想 CPU と 128~512MB のメモリを割り当てた．比較として，機密 VM を用いずにホスト上で機密 Unikernel を動かした場合についても実験を行った．

5.1 機密 Unikernel の監視

機密 Unikernel 内にあるメモリ，CPU，TCP 通信に関する情報を取得する監視機構を実行した．その結果，メモリについては使用可能なメモリのサイズ，使用中のメモリのサイズなどが取得できた．CPU については，ユーザ空間とカーネル空間の CPU 使用時間がそれぞれ取得できた．TCP 通信に関しては，送信元と宛先の IP アドレスとポート番号が取得できた．これらの情報の監視を許可するポリシを設定しなかった場合，監視機構が異常終了した．このことから，ポリシで許可されていないメモリ領域は暗号化されたままであることが分かった．

次に，監視機構が取得したメモリ使用量を機密 Unikernel 内で`/proc/meminfo` から取得したメモリ使用量と比較した．その結果，図 5 のように観測されるメモリ使用量の変化がほぼ一致した．このことから，機密 Unikernel の外で実行される監視機構はリアルタイムに正確なメモリ使用量を把握できることが分かった．同様に，監視機構が取得した CPU 使用時間から計算した CPU 使用率を，機密 Unikernel 内で getrusage システムコールを用いて取得した CPU 使用時間から計算した CPU 使用率と比較した．その結果，図 6 のように観測される CPU 使用率の変化がほぼ一致した．このことから，監視機構はリアルタイムに正確な CPU 使用率を把握できることが分かった．

5.2 オーバヘッド

機密 Unikernel へのメモリ割り当て量を変えながら起動時間を測定した．比較として，従来の機密 Unikernel の起動時間も測定した．その結果，図 7 のように ShadowMonitor による起動時間の増加は最大 0.2%であった．機密 Unikernel の起動時にはシャドウページテーブルが作成されるが，そのオーバヘッドは十分小さいと言える．図 8 は機密 VM

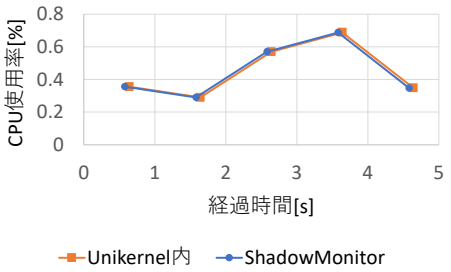

図 6 Unikernel 内外で取得した CPU 使用率

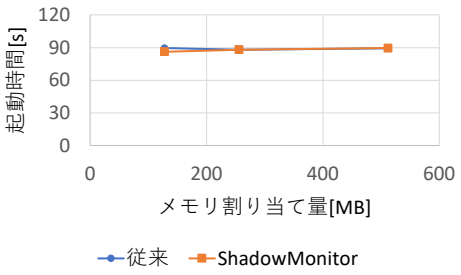

図 7 起動に要した時間 (機密 VM)

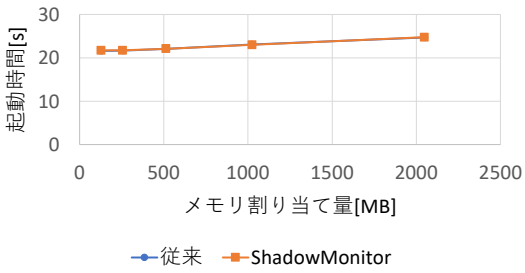

図 8 起動に要した時間 (ホスト)

を用いずにホスト上で機密 Unikernel を動かした場合の起動時間であるが，この場合もオーバヘッドは最大 0.5%と小さかった．

次に，機密 Unikernel のメモリ使用量を変えながら実行時間を測定した．その結果，図 9 のように ShadowMonitor における実行時間の増加は最大 5.6%であった．機密 VM を用いない場合には，図 10 のようにオーバヘッドは最大 2.3%であったため，機密 VM 内で実行することによってオーバヘッドが大きくなったと考えられる．このオーバヘッドはシャドウページテーブルの同期によるものだと考えられるが，詳細については調査中である．

6. 関連研究

SEVmonitor [8] は機密 VM の外への侵入検知システム (IDS) のオフロードを実現している．SEVmonitor は VM 内でエージェントを動作させることにより，仮想ネットワークまたは共有メモリ経由で VM のメモリデータを取得可能にする．エージェントを保護するために，VM 内の監視対象システムをコンテナや内部 VM に隔離し，エージェントを OS やハイパーバイザ内で実行する．IDS も別の機密 VM で安全に実行し，エージェントと暗号通信を行ってメモリデータを取得する．しかし，Unikernel ではエージェントを安全に実行するのが難しい．また，IDS とエージェント間の通信のオーバヘッドも大きい．このオーバヘッドを減らすために，監視対象システムに eBPF プログラムを送り込む eBPFmonitor [12] が提案されているが，Unikernel では eBPF プログラムを動かすのは難しい．

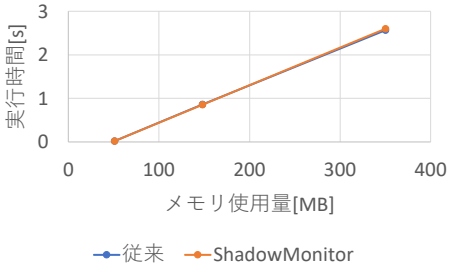

図 9 実行に要した時間 (機密 VM)

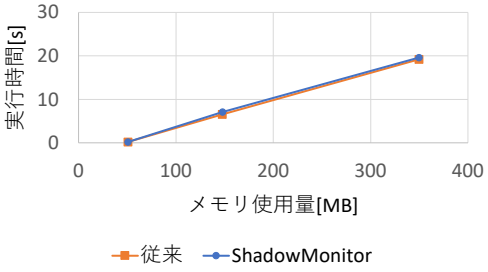

図 10 実行に要した時間 (ホスト)

Ryoan [13] は Google NaCl を用いて Intel SGX のエンクレイヴ内にサンドボックスを作成し，その中で実行されるクラウドサービスを監視する．NaCl を用いることにより，サンドボックス内でのサービスの実行開始時にコードを検査したり，実行時チェックを行ったりすることができる．これにより，安全にサービスの通信履歴を取得したり，サービスのアクセス制限を行ったりすることができる．NaCl のランタイムはエンクレイヴ内で実行されるため，クラウドからの監視の妨害を防ぐことができる．

AccTEE [14] は NaCl の代わりに WebAssembly を用いて SGX エンクレイヴ内に双方向サンドボックスを作成し，その中で実行されるプログラムを監視する．これにより，サンドボックス内のプログラムによるリソース利用情報をサンドボックス外部で安全に記録することができる．記録された情報はエンクレイヴ外部からもサンドボックス内のプログラムからも保護することができる．しかし，Ryoan も AccTEE も監視機構はサンドボックス内のプログラムのデータをすべて参照することができるため，機密情報を盗聴される恐れがある．

SEV-tracker [15] はネストした仮想化 [16] を用いて機密 VM 内で機密 VM を動作させることで，クラウド内で安

全に通信の追跡や制御を行う．ネストした VM の中でクラウドサービスを動作させ，その通信を外側の VM 内で監視する．このようなシステム構成にすることにより，監視機構をクラウドからもクラウドサービスからも保護することができる．SEV-tracker は軽量なハイパーバイザや Unikernel を用いることでネストした仮想化のオーバヘッドを削減しているが，それでもまだオーバヘッドは大きい．

機密 Unikernel に似た概念として機密 Container [17] が提案されている．機密 Container はコンテナに様々な TEE を適用したものである．Kata Container [18] などの VM ベースのコンテナランタイムを用いる場合には VM 向けの TEE を適用することができる．Kata Container は VM 内で汎用 OS の Linux と Kata エージェントを動作させ，Kata エージェント経由でコンテナの監視を行うことができる．しかし，Kata エージェントが正常に動作しなくなると監視が行えなくなる．

7. まとめ

本稿では，AMD SEV-SNP によるメモリ保護を Unikernel がきめ細かく制御し，Unikernel 外部からの監視を可能にする ShadowMonitor を提案した．ShadowMonitor はユーザが指定したポリシに基づいて一部のメモリ領域の暗号化を解除することで，監視機構が VM イントロスペクションを用いて OS データを取得できるようにする．アドレス変換に必要なページテーブルの暗号化は解除できないため，暗号化を行わないシャドウページテーブルを作成する．ShadowMonitor を Nanos と KVMonitor に実装し，Unikernel 内の情報が取得できることを確認した．また，ShadowMonitor によるアプリケーションの性能低下について確認した．

今後の課題は，既存のアプリケーションを機密 Unikernel として実行して監視できるようにすることと，機密 Unikernel 内のアプリケーションの情報も監視できるようにすることである．

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] Madhavapeddy, A., Mortier, R., Rotsos, C., Scott, D., Singh, B., Gazagnaire, T., Smith, S., Hand, S. and Crowcroft, J.: Unikernels: library operating systems for the cloud, *Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems*, pp. 461–472 (2013).
[2] Advanced Micro Devices, Inc.: Secure Encrypted Virtualization API Version 0.24 (2020).
[3] Intel Corporation: Intel Trust Domain Extension (2023).
[4] Google Cloud: Confidential VM overview, https://cloud.google.com/confidential-computing/confidential-vm/docs/confidential-vm-overview.
[5] Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proc. Network and Destributed Systems Security Symp*, pp. 191–206 (2003).
[6] NanoVMs Inc.: Nanos, https://nanos.org/ (2023).
[7] Kourai, K. and Nakamura, K.: Efficient VM Introspection in KVM and Performance Comparison with Xen, *Proc. Pacific Rim Int. Symp. Dependable Computing*, pp. 192–202 (2014).
[8] Nono, T. and Kourai, K.: Secure Monitoring of Confidential VMs with Isolated Agents, *Proceedings of the 18th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2025), Article No.18*, p. 10 (2025).
[9] 瀧口和樹, 光来健一：AMD SEV/-ES/-SNP によるネストした VM の保護方式，第 166 回 OS 研究会 (2025).
[10] Haas, A., Rossberg, A., Schuff, D., Titzer, B., Holman, M., Gohman, D., Wagner, L., Zakai, A. and Bastien, J.: Bringing the Web Up to Speed with WebAssembly, *Proc. ACM SIGPLAN Conf. Programming Language Design and Implementation*, pp. 185–200 (2017).
[11] Ozaki, Y., Kanamoto, S., Yamamoto, H. and Kourai, K.: Detecting System Failures with GPUs and LLVM, *Proc. ACM SIGOPS Asia-Pacific Workshop on Systems* (2019).
[12] 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，第 36 回コンピュータシステム・シンポジウム (ComSys 2024) (2024).
[13] Hunt, T., Zhu, Z., Xu, Y., Peter, S. and Witchel, E.: Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data, *Proc. USENIX Symp. Operating Systems Design and Implementation* (2016).
[14] Goltzsche, D., Nieke, M., Knauth, T. and Kapitza, R.: AccTEE: A WebAssembly-based Two-way Sandbox for Trusted Resource Accounting, *Middleware '19: Proceedings of the 20th International Middleware Conference* (2019).
[15] Ando, N., Takiguchi, K. and Kourai, K.: Secure Privacy Control inside Clouds with AMD SEV and Nested Virtualization, *Proceedings of the 49th IEEE International Conference on Computers, Software, and Applications (COMPSAC 2025)*, pp. 83–88 (2025).
[16] Ben-Yehuda, M., Day, M. D., Dubitzky, Z., Factor, M., N.Har'El, Gordon, A., Liguori, A., Wasserman, O., Yassour, B.-A.: The Turtles Project: Design and Implementation of Nested Virtualization, *Proc. USENIX Conf. Operating Systems Design and Implementation*, pp. 423–436 (2010).
[17] Confidential Containers Contributors: Confidential Containers, https://confidentialcontainers.org/.
[18] OpenInfra Foundation: The speed of containers, the security of VMs, https://katacontainers.io/.