AMD SEVで保護されたUnikernel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 向けの Trusted Execution Environment (TEE) である AMD SEV などを用いて保護することにより，アプリケーションの持つ機密情報の漏洩を防ぐことができる．このような Confidential 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，Confidential 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 のデータを監視することもできない．本稿では，SEV で保護された Confidential Unikernel の監視を Unikernel 自身によるメモリ暗号化の制御により実現するシステム 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を暗号化しないようにすることで，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．SEV においては Unikernel のメモリを解析する際に参照する必要があるページテーブルの暗号化を解除することができないため，ページテーブルを複製して暗号化しないようにした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 の実行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，攻撃も受けにくいという特長がある．その一方，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．そこで，最近のクラウドでは AMD SEV [2] や Intel TDX [3] などの Trusted Execution Environment (TEE) を用いて保護された Confidential VM が提供されている [4]．Confidential VM はメモリを暗号化することによって盗聴を防いでおり，Unikernel も同様に 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 Confidential Unikernel と呼ぶ．

TEE によって保護されていても Confidential Unikernel には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のみ VM のメモリを保護するため，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に動作しないことがある．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，VM のメモリが TEE で保護されているとこの手法を用いることもできない．

そこで本稿では，SEV で保護された Confidential Unikernel の監視を Unikernel 自身によるメモリ暗号化の制御によって実現するシステム 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を暗号化しないようにすることで，VM イントロスペクションを用いて VM 外部から監視に必要なデータを取得できるようにする．Unikernel のメモリを解析する際には 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 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．そのため，ShadowMonitor はページテーブルを複製して暗号化しないようにしたシャドウページテーブルを Unikernel 内に作成する．VM 外の監視機構はシャドウページテーブルを参

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

照することでアドレス変換を行う．

我々は 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．実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，ページテーブルエントリ (PTE) の更新時にシャドウページテーブルの PTE も更新する．シャドウページテーブルに新しいページを割り当てた時には，実ページテーブルの PTE の C ビットをクリアすることでシャドウページテーブルのページの暗号化を解除する．また，Unikernel 内の監視対象データが配置されているページの暗号化も解除する．ShadowMonitor を用いて実験を行い，VM 外から Confidential Unikernel 内の情報が取得できることを確認した．また，シャドウページテーブルを作成するオーバヘッドが小さいことも確認した．

以下，2 章では Confidential Unikernel を監視する上での問題点を述べる．3 章では Confidential Unikernel の監視をメモリ暗号化の制御により実現するシステム 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. Confidential Unikernel の監視

Confidential Unikernel は Confidential 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．Confidential Unikernel に用いることができる TEE の一つである SEV 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 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 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，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 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 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 DoS 攻撃にもより脆弱である．

そのため，Confidential Unikernel では Confidential VM と同様もしくはそれ以上に，異常が発生していないかの監視を行う必要性が高い．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も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直接的に受けてしまう．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．

VM 内のシステムの監視を行うには，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 VM イントロスペクションがよく用いられる．VM イントロスペクションは 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 のページテーブルにアクセスすることで OS データの仮想アドレスを物理アドレスに変換し，さらにその物理アドレスをホストアドレスに変換する．この手法は Unikernel に対しても利用することができる．しかし，VM のメモリが SEV で暗号化されている場合には VM 外の監視機構が VM のメモリにアクセスすることはできないため，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，Confidential 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ない．Confidential VM 内部に VM を作成して監視対象システムを隔離する手法も提案されているが，Unikernel の軽量さが失われる．

3. ShadowMonitor

本稿では，Confidential Unikernel の監視を Unikernel 自身によるメモリ暗号化の制御により実現するシステム ShadowMonitor を提案する．ShadowMonitor のシステム構成を図 1 に示す．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を暗号化しないようにする．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

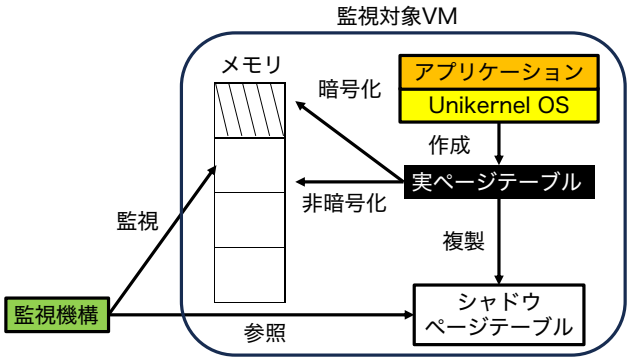

図 1 ShadowMonitor のシステム構成

やアプリケーションの管理情報などが格納されている領域が挙げられる．Unikernel の OS のほとんどの領域は暗号化する必要がないことが予想される．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行う．

暗号化しないようにしたメモリ領域に VM 外の監視機構が 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 のメモリデータの解析には Unikernel のページテーブルを用いて監視対象データのアドレス変換を行う必要がある．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 は VM 内のページテーブルを常に暗号化する．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．

VM 外の監視機構はシャドウページテーブルを参照することで，Confidential 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実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．実ページテーブルについては CPU レジスタからアドレスを取得することができるが，シャドウページテーブルのアドレスは VM 外から取得できる位置には格納されないためである．そこで，ShadowMonitor はシャドウページテーブルを実ページテーブルと連続する領域に配置する．これにより，CPU レジスタから取得した実ページテーブルのアドレスを用いてシャドウページテーブルのアドレスを計算することができる．

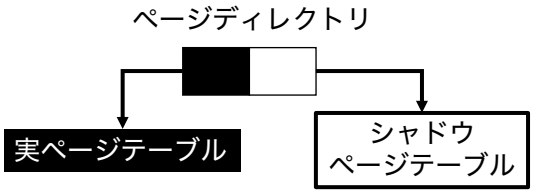

図 2 ページディレクトリの配置

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し続けることができる．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ことによって OS を保護する Unikernel を用いる．もしくは，WebAssembly [9] 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．

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ことによりカーネルモードで動作する OS を保護する．

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 VM に SEV 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．VM 外の監視機構がこのページを特定できるようにするために図 2 のように連続した 2 ページを確保し，1 ページ目を実ページテーブル用に，2 ページ目をシャドウページテーブル用に割り当てる．

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，VM 外の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．

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 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 3 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．

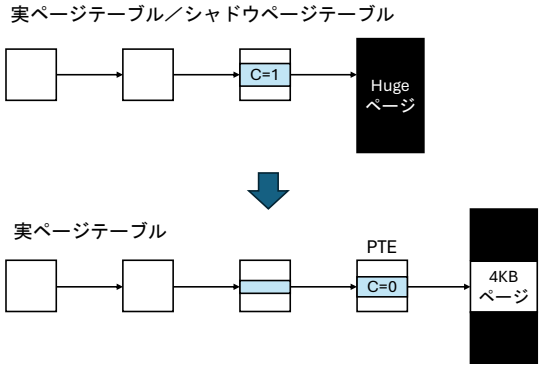

図 3 Huge ページの分割

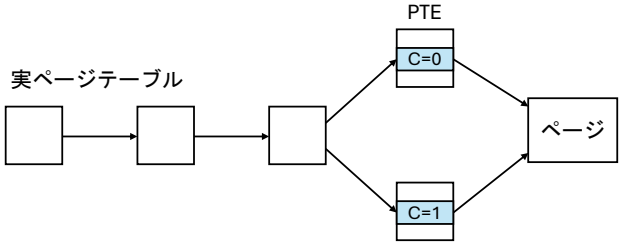

図 4 メモリマッピングの変更

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 4 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

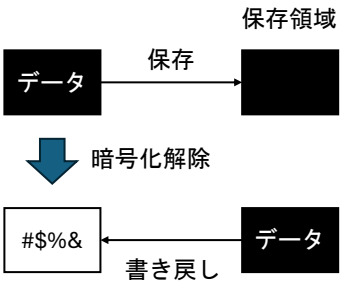

図 5 データが格納されたページの暗号化解除

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 5 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 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 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

OS データの暗号化の解除もシャドウページテーブルのページと同様に，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 ビットをクリアすることで行う．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暗号化されなくなるため，機密情報がある場合にはコンパイル時に別のページに配置されるようにする必要がある．動的に確保されるメモリについては，データの初期化を行う前に暗号化を解除する．ただし，同じページにすでにデータが格納されている場合にはそのデータを保存してから暗号化を解除した後で書き戻す．この場合も暗号化を解除するページ上に機密情報が配置されないようにメモリ割り当てを行う必要がある．

現在の実装では，プロセス構造体に格納されている情報を暗号化しないようにしている．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．

4.4 OS データの監視

LLView [10] 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 を用いて VM 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．まず，QEMU に QMP コマンドを送信して 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する．次に，このアドレスに 4KB を加えることでシャドウページテーブルの物理アドレスを計算する．そして，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．得られた物理アドレスをホストアドレスに変換し，メモリマップされた VM のメモリファイルにアクセスすることで，暗号化されていない OS データを取得することができる．

4.5 Nanos の SEV 対応

Nanos を実行する VM のメモリを暗号化できるようにするために Nanos の SEV 対応を行った．Nanos の起動時に SEV を利用できるかどうかを確認するために，CPU から SEV の情報を取得する．まず，CPUID 命令を用いて CPU が SEV をサポートしているかどうかを調べる．SEV がサポートされていれば，RDMSR 命令を用いてモデル固有レジスタから SEV が有効かどうかの情報を取得する．SEV が有効であれば，PTE の何ビット目が C ビットとして使われているかの情報を取得する．この情報を用いて，PTE を作成したり更新したりする際には基本的に C ビットを 1 にしてページが暗号化されるようにする．

例外として，デバイスドライバが QEMU との間のやりとりに用いるバウンスバッファは暗号化しないようにする．デバイスドライバが用いるバッファは QEMU からアクセスする必要があるが，バッファが SEV で暗号化されているとアクセスできないためである．デバイスへの書き込み時には，SEV によって暗号化されているバッファのデータをバウンスバッファにコピーしてから書き込み処理を行

```
nisimuray@sub6:proc$ sudo ./proc
pid: 2
total: 121053184
allocated: 55472128
nisimuray@sub6:proc$ sudo ./proc
pid: 2
total: 121053184
allocated: 66355200
```

図 6 取得できた OS データ

う．デバイスからの読み込み時には，バウンスバッファに読み込まれたデータをバッファにコピーしてから読み込み処理を行う．

SEV に対応した Nanos が 2023 年 11 月下旬にリリースされたが，我々はそれ以前に独自に SEV 対応を行った．SEV の拡張である SEV-SNP にも対応済みであるが，ShadowMonitor がまだ SEV-SNP に対応できていないため本稿では説明を省略する．

5. 実験

ShadowMonitor を用いて SEV で保護された VM 内で動作している Unikernel のデータが取得できることを確認し，アプリケーションの実行時間への影響を調べた．実験には AMD EPYC 7443P の CPU，256GiB のメモリ，2TB の SATA HDD を搭載したマシンを用いた．OS として Linux 5.15, 仮想化ソフトウェアとして QEMU-KVM 7.1.0 を動作させた．VM には 128MB のメモリを割り当て，Unikernel として Nanos 0.1.46 を動作させた．

5.1 動作確認

メモリの確保のみを繰り返す Unikernel を実行し，OS データを取得する監視プログラムを実行した．実行結果を図 6 に示す．OS データとしてプロセス ID，Unikernel が使用可能なメモリのサイズ，Unikernel が使用中のメモリのサイズなどが取得できることを確認した．また，監視プログラムを繰り返し実行すると使用中のメモリが増えていく様子が観測できた．最終的には Unikernel が異常終了したため，ShadowMonitor を用いた監視によって異常終了する前に異常を検知できることが分かった．

5.2 オーバヘッド

シャドウページテーブルを作成するオーバヘッドを調べるために，Unikernel の実行にかかる時間を測定した．この実験では作成されるページテーブルのサイズの影響を調べるために，メモリ使用量が少ない Unikernel と malloc に失敗するまでメモリを確保してメモリを 24MB 多く使用する Unikernel を用いた．比較のために，ページテーブルの複製を行わない従来の Nanos を用いた場合についても測定を行った．測定を 50 回行った時の実験結果を図 7 と

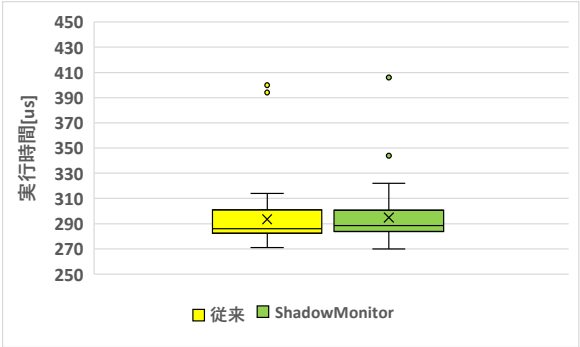

図 7 メモリ使用量が少ない Unikernel の実行時間

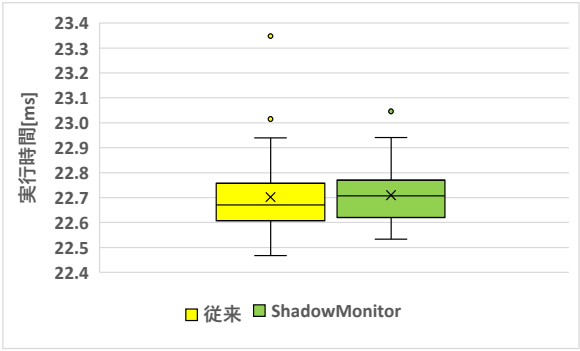

図 8 メモリ使用量が多い Unikernel の実行時間

図 8 に示す．メモリ使用量が少ない Unikernel でのオーバヘッドは 0.7%，メモリ使用量が多い Unikernel では 0.04% であった．後者の方がページテーブルのサイズは大きくなったが，実行に時間がかかるようになったため，オーバヘッドはむしろ小さくなった．これより，ShadowMonitor のオーバヘッドは十分に小さいといえる．

6. 関連研究

SEVmonitor [8] は Confidential 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 Confidential 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 [11] 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．

Ryoan [12] は Google NaCl [13] 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 Confidential VM 内で Confidential 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．

Confidential Unikernel に似た概念として Confidential Container [17] が提案されている．Confidential 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，Confidential Unikernel の監視を Unikernel 自身によるメモリ暗号化の制御により実現するシステム ShadowMonitor を提案した．ShadowMonitor は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 外の監視機構が 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，Nanos のカーネル内の情報が取得できることを確認した．また，シャドウページテーブルの作成によるアプリケーションの性能低下が 0.7% 以下であることも確認できた．

今後の課題は，CPU のレジスタも暗号化される SEV-SNP に対応することである．CR3 レジスタが暗号化されるとシャドウページテーブルのアドレスを計算するために必要な実ページテーブルのアドレスが取得できなくなるため，別の手段で取得できるようにする必要がある．また，監視する必要のある OS データと機密情報が同一のページ上にあるようであれば，別々のページに配置されるようにコンパイル時および実行時に調整でき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．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] 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 (2024).
[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] NanosVMs 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] 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).
[9] 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).
[10] 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).
[11] 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).
[12] 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).
[13] Google, Inc.: Native Client, https://developer.chrome.com/docs/native-client/ (2016).

[14] Goltzsche, D., Nieke, M., Knauth, T. and Kapitza, R.: AccTEE: A WebAssembly-based Two-way Sandbox for Trusted Resource Accounting, *Proc. Pacific Rim Int. Symp. Dependable Computing* (2019).
[15] 安東尚哉, 瀧口和樹, 光来健一: AMD SEV を用いてネストした VM を保護することによる安全な通信の追跡・制御, 第 104 回 CSEC 研究会 (2024).
[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/ (2024).
[18] OpenInfra Foundation: The speed of containers, the security of VMs, https://katacontainers.io/ (2024).