AMD SEV とネストした仮想化を用いた安全な通信履歴の取得

安東 尚哉[1]　光来 健一[1]

概要：近年，パブリッククラウドからのパーソナルデータの漏洩が問題となっている．クラウドは複雑なサービスを提供するために複数のサービスを連携させることが多くなっており，パーソナルデータが様々なサービスに転送されるようになっている．一方，クラウド内のデータ流はユーザには非公開であるため，ユーザはパーソナルデータの漏洩を把握することができない．この問題を解決するためにはクラウド内にプライバシ制御機構が必要となるが，クラウドが提供するプライバシ制御機構を完全に信頼することはできない．本稿では，AMD SEV を用いてクラウド内のデータ流を安全に追跡・制御するシステム SEV-tracker を提案する．SEV-tracker はネストした仮想化を用いて，クラウドの仮想マシン（VM）内でユーザが送り込んだハイパーバイザを実行し，その上のユーザ VM で動作するクラウドサービスの通信履歴を記録する．SEV を用いてそれぞれの VM のメモリを保護することにより，クラウドがユーザ・ハイパーバイザを攻撃したり，ユーザ・ハイパーバイザがクラウドサービスを攻撃したりすることを相互に防ぐ．SEV-tracker を軽量な BitVisor および Unikraft を用いて実装し，クラウドサービスの性能や通信履歴の取得性能を調べた．

1. はじめに

近年，不正アクセスや設定ミスによるパブリッククラウドからのパーソナルデータの漏洩が問題となっている．2019 年 7 月には Web ファイアウォールの設定ミスにより，米金融大手の Capital One から 1 億人以上の個人情報が流出した．日本でも，2020 年 12 月にクラウドサービスの設定ミスにより楽天から約 148 万件の個人情報が流出している．その原因の一つとして，クラウドが年々，複雑なサービスを提供するようになっていることが挙げられる．マイクロサービスやマルチクラウドを用いて複数のサービスを連携させることが多くなっているため，パーソナルデータがクラウド内，さらにはクラウド間で様々なサービスに転送されるようになっている．

クラウドが内部でどのようにデータを扱っているかの詳細についてはユーザに公開されていないため，パーソナルデータがどのクラウドサービスに転送されているのかもわからないことが多い．そのため，ユーザは自分のパーソナルデータが意図しないサービスに転送されたり，漏洩したりしていてもそのことを把握することができない．この問題を解決するためにはユーザが自分のデータの流れを追跡・制御することができるプライバシ制御機構がクラウド内に必要となる．このような機能があれば，自分のデータの流通範囲や保存先を把握することができ，データの流れを制御することで情報漏洩を未然に防ぐことができる．しかし，ユーザからはクラウドが提供するプライバシ制御機構を完全には信頼することができない．

本稿では，AMD SEV [1] を用いてクラウド内のデータ流を安全に追跡・制御するシステム SEV-tracker を提案する．SEV-tracker はネストした仮想化 [2] を用いて，クラウドが提供する仮想マシン（VM）内でユーザが送り込んだハイパーバイザを実行し，その中でプライバシ制御機構を動作させる．そして，その上に作成したユーザ VM でクラウドサービスを動作させ，クラウドサービスの通信等を捕捉してデータ流の追跡・制御を行う．SEV を用いてクラウド VM とユーザ VM のメモリを保護することで，クラウドがユーザ・ハイパーバイザを攻撃してプライバシ制御機構を無効化することや，ユーザ・ハイパーバイザがクラウドサービスを攻撃することを相互に防ぐ．

ネストした仮想化を用いてクラウド VM 内でユーザ VM を実行するオーバヘッドを削減するために，軽量なハイパーバイザとゲスト OS を用いて SEV-tracker を実装した．クラウド VM 内で動作するユーザ・ハイパーバイザとして軽量な仮想化を行う BitVisor [3] を用い，ユーザ VM による通信を記録できるようにした．BitVisor 内に lwIP の Raw

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

API を用いてサーバを実装し，クライアントと BitVisor および，BitVisor 間でデータ追跡に必要な通信履歴の取得を行えるようにした．また，ユーザ VM 内でクラウドサービスを実行するゲスト OS として，必要最小限の機能のみを提供するライブラリ OS である Unikraft を用いた．ユーザ VM に対して SEV を適用できるようにするために，UEFI を用いて Unikraft を起動できるようにし，BitVisor 上で動作させるための修正も行った．

SEV-tracker を用いてクラウド VM 内で BitVisor と Unikraft を起動する実験を行った結果，起動時間は 4 秒程度となり，標準的な KVM と Linux を起動する場合に比べて約 7 倍高速であることがわかった．必要なメモリ量も 4 分の 1 で済むことがわかった．また，クラウドサービスによる通信の履歴をクライアントが取得できることを確認した．再帰的に複数のユーザ・ハイパーバイザから通信履歴を取得した場合，クラウド VM の数に応じた時間がかかることがわかった．

以下，2 章ではクラウドにおけるプライバシ制御機構の必要性について述べる．3 章では SEV を用いてクラウド内でプライバシ制御機構を安全に動作させてデータ流を追跡・制御するシステム SEV-tracker を提案する．4 章では SEV-tracker の実装について説明する．5 章では SEV-tracker の性能や通信履歴の取得について調べた実験について述べる．6 章で関連研究について述べ，7 章で本論文をまとめる．

2. プライバシ制御機構の必要性

クラウドの普及率は年々上昇しており，それに伴ってクラウドが大量のパーソナルデータを扱うようになっている．パーソナルデータとは個人に関する情報全般のことであり，個人を特定，識別することができる個人情報だけでなく，サービスの利用情報なども含まれる．パーソナルデータを扱うクラウドサービスが増え，クラウドサービスを利用するユーザが増えた結果，クラウドからのパーソナルデータの大規模漏洩が問題となっている．

その一つの原因として，クラウドが複雑なサービスを提供するようになっていることが挙げられる．最近のクラウドでは，マイクロサービスアーキテクチャなどのように複数のサービスを連携させて動作させるのが一般的になっている．マイクロサービスは複雑なサービスを複数の小さなサービスに分割してソフトウェアを開発する手法である．サービス間でデータをやり取りしながら動作するため，パーソナルデータも様々なサービスに転送されることになる．また，複数のクラウドによって提供されるサービスを利用するマルチクラウドを用いる場合には，パーソナルデータが一つのクラウド内だけでなく，他のクラウドにも転送される．

一方で，パブリッククラウドにおいてはクラウドサービスが扱うデータの流れは基本的に非公開となっている．ユーザは利用したサービスがパーソナルデータを他のどのサービスに転送したのかを追跡したり，世界各地にあるデータセンタのどこにパーソナルデータが保存されているのかを特定したりすることはできないことが多い．また，ユーザがパーソナルデータの流れを制御するのも難しいことが多い．プライベートクラウドやガバメントクラウドなどではパーソナルデータの流通範囲や保存先を限定することができるが，パブリッククラウドと比べてコストの上昇は避けられない．

このような問題を解決するためには，パブリッククラウドにおいてユーザが自分のパーソナルデータを追跡・制御できるようなプライバシ制御機構が必要である．クラウド内でプライバシ制御機構を動作させることで，ユーザは自分のパーソナルデータがどのクラウドサービスに転送され，どのデータセンタに保存されたかを把握することができるようになる．ユーザは必要に応じて自分のパーソナルデータの流通範囲や保存先を調べることにより，安心してクラウドサービスを利用できるようになる．また，パーソナルデータの流通範囲を制限することで情報漏洩を未然に防ぐこともできる．ただし，それによって利用できるサービスが制限されたり，サービスの利用料が高くなったりする可能性はある．

しかし，クラウドがプライバシ制御機構を提供してもユーザはそれを完全には信用することができない．パーソナルデータが正しく追跡・制御されていることをユーザからは確認できないためである．クラウドはプライバシ制御機構をバイパスして，データ収集サーバにパーソナルデータを転送している可能性がある．もし，クラウド内に内部犯がいた場合には，外部からは正しく追跡・制御できているように見えても，実際は情報が漏洩している可能性もある．実際に，サイバー犯罪の 28%が内部犯によるものであるという調査結果 [4] や管理者の 35%が機密情報を盗み見たことがあるという調査結果 [5] もある．

3. SEV-tracker

本稿では，AMD SEV [1] を用いてユーザがクラウド内のデータ流を安全に追跡・制御することを可能にするシステム SEV-tracker を提案する．SEV-tracker のシステム構成は図 1 のようになる．クラウド内でプライバシ制御機構が正しく動作することを保証するために，SEV-tracker はネストした仮想化 [2] を用いて，ユーザが送り込んだハイパーバイザをクラウドの VM 内で実行する．このような構成にすることにより，クラウド VM がネストした仮想化と SEV をサポートしていれば，既存のクラウドに適用することができる．送り込んだユーザ・ハイパーバイザ上でユーザ VM を作成し，ユーザ VM 内でクラウドサービスを動作させる．これにより，クラウドサービスの通信やディス

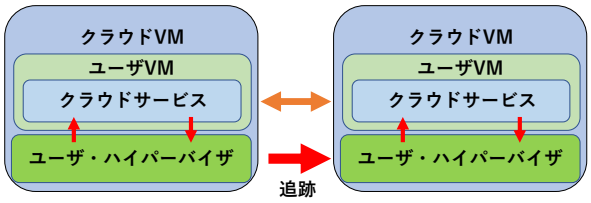

図 1 SEV-tracker のシステム構成

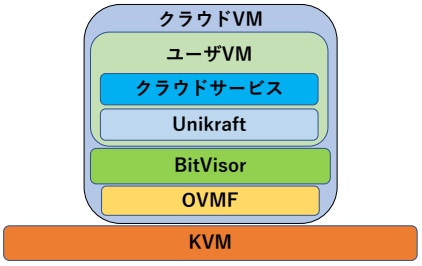

図 2 BitVisor と Unikraft を用いたシステム構成

クアクセスなどをユーザ・ハイパーバイザ内のプライバシ制御機構が捕捉して追跡・制御を行うことができる．

本研究では，クラウドとユーザが互いに信頼できないという脅威モデルを考える．そのため，クラウドとユーザ・ハイパーバイザは相互に保護を行うことが必要となる．通常，クラウドは VM 内で動作するユーザ・ハイパーバイザよりも高い権限を持つため，ユーザ・ハイパーバイザはクラウドからの攻撃を受ける可能性がある．その場合には，プライバシ制御機構が無効化されたり，データの追跡情報を改竄されたりして，正常にデータの追跡・制御ができなくなる恐れがある．また，ユーザ・ハイパーバイザはユーザ VM 内で動作するクラウドサービスより高い権限を持つため，ユーザ・ハイパーバイザがクラウドサービスを攻撃することができる．この場合，ユーザがクラウドサービスに関する情報を盗んだり改竄したりすることができてしまう．

そこで，SEV-tracker は SEV を用いることで相互保護を実現する．SEV は AMD 製 CPU が提供する VM のメモリ暗号化機能であり，VM がメモリにデータを書き込む際に暗号化を行い，読み込む際に復号化を行う．メモリ暗号化のための鍵は AMD セキュアプロセッサで生成・管理されるため，ハイパーバイザから VM を保護することができる．SEV-ES ではさらに VM のレジスタも暗号化され，SEV-SNP では VM のメモリの整合性も保証されるが，本稿ではこれらを含めて SEV と呼ぶ．ユーザ・ハイパーバイザが動作するクラウド VM のメモリを SEV で保護することにより，クラウドから攻撃されたとしてもデータの改竄や窃取を防ぐことができる．また，クラウドサービスが動作するユーザ VM のメモリも SEV で保護することで，ユーザ・ハイパーバイザからの攻撃を防ぐことができる．クラウド VM とユーザ VM でそれぞれ正しいユーザ・ハイパーバイザとクラウドサービスが実行されていることは，SEV のリモートアテステーションを用いて確認することができる．

SEV-tracker はネストした仮想化を用いるため，クラウドサービスの実行オーバヘッドが大きくなる．このオーバヘッドを削減するために，SEV-tracker はユーザ・ハイパーバイザとしてできるだけ軽量なハイパーバイザを用いる．通常，ハイパーバイザは複数の VM をサポートしているが，ユーザ・ハイパーバイザは一つのユーザ VM のみをサポートすれば十分である．複数の VM を動作させるのに必要な機能を省くことによって，ハイパーバイザを軽量化し，オーバヘッドを削減する．また，ハイパーバイザはデバイスを仮想化して VM に提供するが，ユーザ・ハイパーバイザはデータの追跡・制御を行う必要のないデバイスは仮想化する必要がない．パススルー機能を用いてクラウド VM の仮想デバイスをユーザ VM にそのまま見せることで軽量化を実現する．

さらに，ユーザ VM 内のクラウドサービスには軽量なライブラリ OS をゲスト OS として使わせる．ライブラリ OS とは OS の機能をライブラリとして提供し，アプリケーションにリンクして利用することができる OS である．アプリケーションをコンパイルする際に必要な機能のみをリンクすることで汎用 OS を使う場合よりも高速に動作し，メモリ使用量を抑えることができる．また，ライブラリ OS は最小限の初期化しか行わないため，クラウドサービスの起動を速くすることもできる．

SEV-tracker はクラウドサービスのすべての通信を監視することにより，データ流の追跡を可能にする．ユーザ VM 内のクラウドサービスが送受信を行うたびに，ユーザ・ハイパーバイザ内のプライバシ制御機構がパケットを捕捉して解析し，通信情報を記録する．ユーザがユーザ・ハイパーバイザに要求を送ると，その上で実行されているクラウドサービスの通信履歴を返す．さらに通信先でもユーザ・ハイパーバイザが動作している場合には，プライバシ制御機構から再帰的に要求を送ってそのクラウドサービスの通信履歴を取得する．この時，クラウドサービスの通信と同じ経路が使われるため，すべてのユーザ・ハイパーバイザと通信することができる．

4. 実装

我々はユーザ・ハイパーバイザとして BitVisor [3] を用い，ユーザ VM のゲスト OS として Unikraft[6] を用いて SEV-tracker を実装した．標準の KVM 上に作成された VM をクラウド VM として用いた．KVM では SEV を有効にするために BIOS の後継である UEFI を使用する必要があるため，オープンな UEFI ファームウェアである OVMF [7] を用いた．システムの構成を図 2 に示す．

4.1 BitVisor と Unikraft

BitVisor は一般的なハイパーバイザとは異なり，VM を一つしかサポートしていない．それによって複数の VM を動作させるのに必要な機能を省いたため，ハイパーバイザ自体が軽量になっている．また，デバイスは基本的にそのまま VM に見せ，仮想化する必要のあるデバイスについてのみハイパーバイザで最小限の処理を行う．このような方式を準パススルーと呼び，仮想化したデバイスを処理するドライバを準パススルードライバと呼ぶ．BitVisor にはネットワークやストレージなどの準パススルードライバがある．さらに，BitVisor には lwIP [8] を用いた TCP/IP 機能が内包されているため，ハイパーバイザ単体でネットワーク通信を行うことができる．BitVisor は UEFI での起動に対応している．

Unikraft はライブラリ OS を用いて，単一アドレス空間で動作する Unikernel と呼ばれるアプリケーションを容易に開発できるようにしたシステムである．開発者はアプリケーションの実行に必要な OS の機能を提供するライブラリを選択してコンパイルする．KVM や Xen など複数のプラットフォームをサポートしており，プラットフォーム固有の開発を行うことなくそれぞれのプラットフォームに対応したイメージを作成することができる．クラウドサービスで一般的な単一目的のアプリケーションを作成し，ハイパーバイザ上の VM 内で汎用 OS を介さずに直接，実行される．Unikraft は BIOS での起動のみに対応している．

4.2 Unikraft の UEFI 対応

SEV を用いるためには Unikraft を UEFI で起動できるようにする必要があるため，Unikraft に UEFI 対応を行った．UEFI 上で動くアプリケーションは PE バイナリとなっている．一方，従来の Unikraft ではイメージが ELF バイナリとして生成されるため，そのままでは UEFI アプリケーションとして扱うことはできない．UEFI 対応を行うにあたって文献 [9] を参考にしたが，この文献が対象としている Unikraft のバージョンは 0.3 であり，本研究で用いたバージョン 0.5 では対応が必要な箇所の実装が一部変更されていた．

Unikraft アプリケーションのロードの流れを図 3 に示す．UEFI アプリケーションがロードされるメモリアドレスは UEFI が決定するため，UEFI アプリケーションは再配置可能である必要がある．そこで，Unikraft のソースコードを位置独立コードとしてコンパイルし，生成された ELF バイナリから ELF ヘッダを削除する．従来の Unikraft は ELF ヘッダの下にマルチブートヘッダを配置しているが，その代わりに PE ヘッダを配置するようにする．そして，コンパイル時に ELF バイナリのシンボル情報からエントリポイントのアドレスやバイナリサイズを取得して PE ヘッダに格納する．これによって，バイナリの先頭に PE ヘッダが配置されるため，UEFI アプリケーションとして扱うことができる．

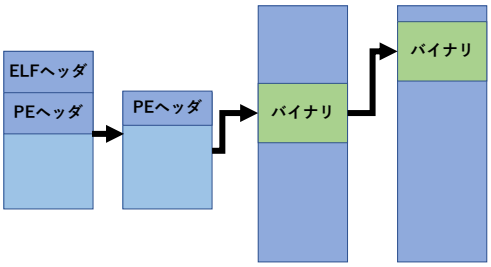

図 3 Unikraft アプリケーションのロード

Unikraft の起動時には，従来の Unikraft が配置されるアドレスである 0x100000 に自分自身を再配置する．まず，UEFI のブートサービスが提供する GetMemoryMap 関数を用いてメモリマップを取得し，再配置先が空き領域であることを確認する．空き領域であればブートサービスの AllocatePages 関数を用いて再配置先にメモリを確保し，CopyMem 関数を用いて Unikraft のコードとデータをコピーする．

従来の Unikraft は起動時に PAE を有効にしたり，プロテクトモードからロングモードに移行したりしているが，UEFI で起動する際にはこれらの処理は行わない．また，マルチブートの機能を用いて行われているコマンドラインの取得やスタックとヒープの設定などは UEFI のブートサービスを用いて行う．その後で，ExitBootServices 関数を呼び出してブートサービスの利用を終了し，OS が UEFI から独立して動作するようにする．最後に，再配置先の関数を呼び出すことで従来の Unikraft と同じメモリ領域を使って実行を行う．

Unikraft と UEFI では関数の呼び出し規約が異なるため，相互に関数を呼び出す際には変換を行う．Unikraft では System V AMD64 ABI に従った呼び出し規約が用いられているのに対し，UEFI では Microsoft x64 呼び出し規約が採用されている．具体的には，Unikraft では関数の引数は RDI，RSI，... の順番でレジスタに格納されるが，UEFI では RCX，RDX，... の順番で格納される．そこで，UEFI が Unikraft のエントリポイントを呼び出す際には，Unikraft の呼び出し規約に従うようにレジスタに引数を代入し直してから関数を呼び出す．逆に，Unikraft が UEFI のブートサービスの関数を呼び出す際には逆の変換を行う．

4.3 BitVisor 上での Unikraft の実行

Unikraft は仮想 NIC として virtio-net のみに対応している．クラウド VM の virtio-net をパススルーでそのままユーザ VM に見せることもできるが，SEV-tracker では BitVisor において Unikraft の通信を捕捉する必要があるため，BitVisor の準パススルーを用いる．BitVisor の準パススルードライバである virtio-net ドライバは，ゲスト OS

が PCI の I/O ポートに書き込んだ際にハンドラの登録を行う．しかし，Linux とは異なり，Unikraft ではこのような書き込みが行われないため，virtio-net が利用できるようにならない．そこで，BitVisor の virtio-net ドライバの初期化を行う際にハンドラの登録を行う．

また，Unikraft はレガシー PCI デバイスとして動作する virtio-net の初期化時にデバイスのリセットを行う．しかし，BitVisor がデバイスの状態をリセット完了の状態に変更しないため，Unikraft はリセット完了を待って無限ループに陥る．デバイスのリセットは正常に行われているため，Unikraft においてリセットの完了を待たないようにすることで問題を回避する．

Unikraft を Bitvisor 上で動作させると VGA コンソールの初期化を行う際に頻繁にクラッシュする．調査の結果，Unikraft がコンソールをクリアするために VRAM にスペースを書き込んでいる途中でクラッシュすることがわかった．Unikraft をクラウドで動作させる際には VGA コンソールは不要であるため，コンソールをクリアしないようにしてクラッシュを回避する．

4.4 UEFI シェルを用いた自動起動

SEV-tracker は UEFI シェルを利用して，クラウド VM 内で BitVisor と Unikraft を自動起動する．UEFI シェルはディスク上に startup.nsh というファイルがある場合には，このスクリプトに書かれたコマンドを順番に実行する．SEV-tracker はクラウドのディスク上の特定のディレクトリをクラウド VM のディスクとして用いるため，このディレクトリにスクリプトファイルを配置する．このスクリプトはまず，BitVisor のローダを実行し，BitVisor をロードする．BitVisor がユーザ VM を作成すると制御が UEFI シェルに戻ってくるため，続けて UEFI アプリケーションとして Unikraft アプリケーションを実行する．このようにして，BitVisor 上のユーザ VM 内で動作する Unikraft アプリケーションを起動する．

デフォルトの設定では UEFI シェルは起動後 5 秒経過しないと startup.nsh を実行しないため，即座に実行されるように設定を変更する．UEFI シェルも UEFI アプリケーションの一つであり，startup.nsh を実行するまでの遅延を引数で指定して実行することができる．また，バージョン情報やデバイスマッピングの表示を抑制することもできる．引数の文字列は UCS-2 である必要があるため，文字列をワイド文字の配列に格納して出力するプログラムを作成した．ワイド文字を 2 バイトとして扱うようにこのプログラムをコンパイルして実行し，UCS-2 の文字列をファイルに格納する．そして，UEFI シェルの bcfg コマンドを用いてファイルの内容を NVRAM に書き込むことにより，UEFI シェルに渡される引数を設定する．

Unikraft アプリケーションを起動する際に引数を渡すことによって割り当てる IP アドレスを指定することができる．Unikraft はデフォルトでは DHCP で IP アドレスを取得するが，IP アドレスとネットマスク，ゲートウェイを指定することでその遅延を減らすことができる．

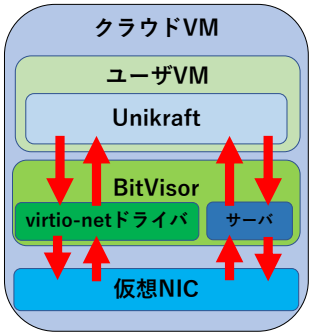

図 4 BitVisor での通信の捕捉

4.5 ネストした仮想化の SEV 対応

KVM は SEV によって保護された VM 内でのハイパーバイザ実行には対応していない．クラウド VM 内で BitVisor を実行できるようにするためには，KVM が仮想化支援機構の SEV を BitVisor のためにエミュレートする必要がある．しかし，BitVisor がユーザ VM を管理するために用いる VMCB もメモリ上にあるため SEV によって暗号化され，KVM がアクセスすることはできない．

一方，クラウド VM のページテーブルの C ビットをクリアして VMCB などのデータが暗号化されないようにすることで，SEV で保護されたクラウド VM 内でもハイパーバイザが動作可能であることが報告されている [10]．クラウドがこれらのデータにアクセスできることでセキュリティが低下する可能性はあるが，クラウド VM 内で BitVisor を動作させることは可能であると考えられる．現在のところ，この機能については SEV-tracker では未実装である．

4.6 BitVisor での通信の捕捉

SEV-tracker は図 4 のように，BitVisor の virtio-net ドライバを用いて Unikraft によるすべての通信を捕捉する．後述するように，SEV-tracker は Unikraft が最終的に使う仮想 NIC と同じ仮想 NIC を用いてクライアント等と通信を行うため，BitVisor の ippass モジュールを用いる．ユーザ VM 内の Unikraft が送信したパケットは BitVisor の virtio-net ドライバによって受信され，クラウド VM の仮想 NIC に送信される．逆に，クラウド VM の仮想 NIC から受信したパケットは virtio-net ドライバに渡され，ユーザ VM に送信される．virtio-net ドライバは捕捉したパケットを解析して IP パケットの場合には送信元アドレスと宛先アドレスをバッファに格納する．ただし，すでに格納されているアドレスの組の場合には格納しない．

4.7 通信履歴の送信

BitVisor は lwIP の Raw API を用いてクライアントおよび他のクラウド VM で動作する BitVisor との通信を行う．lwIP はオープンソースの TCP/IP のプロトコルスタックの実装であり，組込みシステム向けに設計されている．Raw API ではソケットを用いることができず，コールバック関数を用いて通信処理をつなげていく．BitVisor 内のサーバは初期化の際にクライアントから接続された時に呼び出されるコールバック関数を登録する．このコールバック関数では，クライアントから要求を受信した時に呼び出されるコールバック関数を登録する．そのコールバック関数の中でクライアントに通信履歴を送信する．

自身の通信履歴の中に通信先として BitVisor が動作しているクラウド VM の IP アドレスがあった場合には，BitVisor は再帰的に通信を行って通信先の BitVisor の持つ通信履歴を取得する．この場合，クライアントからの要求の受信時に呼び出されるコールバック関数の中で他の BitVisor に要求を送信し，応答を受信した時に呼び出されるコールバック関数を登録する．そのコールバック関数の中で必要であればさらに別の BitVisor とも通信を行う．最終的に，自身の通信履歴と他の BitVisor から受信した通信履歴とをまとめてクライアントに送信する．

5. 実験

SEV-tracker におけるクラウドサービスの性能を調べるために，クラウド VM 内でユーザ・ハイパーバイザを用いてクラウドサービスを起動する時間を測定し，その際に必要となる最小メモリ量について調べた．また，クラウドサービスのベンチマーク性能についても測定した．次に，SEV-tracker を用いてクラウドサービスの通信履歴を取得できることを確認し，取得時間を測定した．本実験では SEV を用いずに VM を作成して実験を行った．実験には**表 1** のマシンを用い，クラウド VM とユーザ VM にはそれぞれ**表 2** と**表 3** の VM を用いた．比較対象として，KVM をユーザ・ハイパーバイザ，Linux をゲスト OS とした場合についても実験を行った．

表 1 ホストマシンの構成

CPU	AMD EPYC 7402P
メモリ	256GB
NIC	10GbE
OS	Linux 5.4
ハイパーバイザ	QEMU-KVM 4.2.1

5.1 クラウドサービスの起動時間

SEV-tracker を用いてクラウドサービスを起動する時間を測定した．起動時間はクラウド VM を起動し始めてから，ユーザ VM 内で Unikraft アプリケーションの main 関数が呼び出されるまでの時間とした．比較として KVM と Linux を用いた場合の起動時間は，Linux のネットワークサービスが利用できるようになるまでの時間とした．また，クラウド VM 内で直接，Unikraft または Linux だけを起動したときの起動時間についても測定した．

表 2 クラウド VM の構成

	SEV-tracker	比較対象
仮想 CPU 数	1	1
メモリ	512MB	512MB
OS	なし	Linux 5.4
ユーザ・ハイパーバイザ	BitVisor	KVM

表 3 ユーザ VM の構成

	SEV-tracker	比較対象
仮想 CPU 数	1	1
メモリ	512MB	512MB
ゲスト OS	Unikraft 0.5	Linux5.4

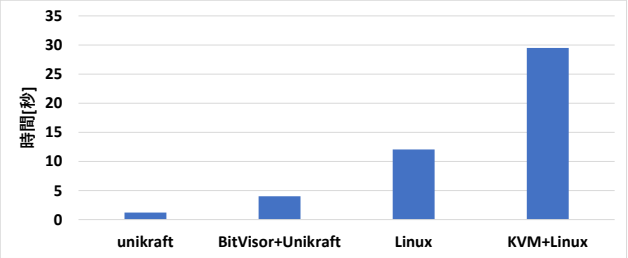

図 5 クラウドサービスの起動時間の比較

それぞれの起動時間を図 5 に示す．クラウド VM 内で BitVisor と Unikraft を起動した場合には，KVM と Linux を起動した場合に比べて 7.3 倍高速であることがわかった．UEFI の機能では BitVisor だけの起動時間を正確に測定することはできなかったが，3 秒程度で起動することが確認できた．KVM を起動するためにはまず，クラウド VM 内で Linux を起動する必要があるため，ユーザ・ハイパーバイザの起動時間の大きな差となった．一方，Unikraft の起動時間は 1 秒程度であり，相対的に BitVisor の起動に時間がかかっていることがわかった．

SEV-tracker を用いずにクラウド VM 内で Linux を単体で起動した場合には 12 秒かかっており，BitVisor と Unikraft を用いる方が 3.0 倍高速であった．本実験ではほぼ何もしない Unikraft アプリケーションを用いたため，最小限の OS 機能の初期化のみで済んだ．実際のクラウドサービスでは利用する OS 機能に合わせた初期化が必要になるため，もう少し時間がかかると思われる．

5.2 クラウドサービスに必要な最小のメモリ量

SEV-tracker を用いてクラウドサービスを起動するために必要な最小のメモリ量を調べる実験を行った．本実験では，クラウド VM に割り当てるメモリ量を減らしながら，クラウドサービスがどこまで起動できるかを調べ

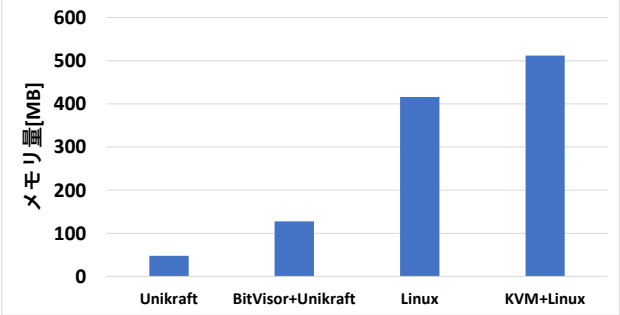

図 6 クラウドサービスの起動に必要な最小メモリ量

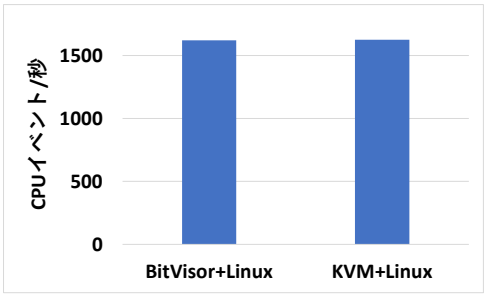

図 7 クラウドサービスの CPU 性能

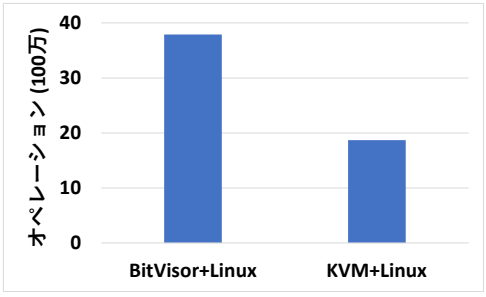

図 8 クラウドサービスのメモリアクセス性能

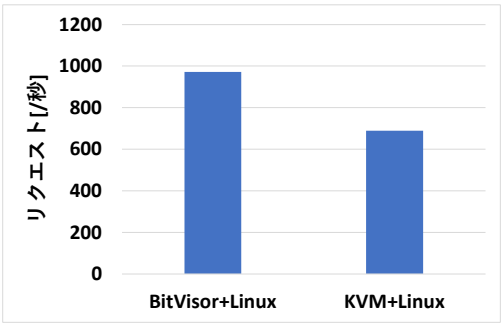

図 9 ウェブサーバのスループット

た．起動に必要な最小メモリ量を図 6 に示す．クラウド VM 内で BitVisor と Unikraft を起動した場合，128MB までは正常に動作したが，それ以下に減らすと BitVisor が”AllocatePages failed”というエラーを出力し，起動に失敗した．一方，クラウド VM 内で KVM と Linux を起動した場合，512MB を下回ると Linux カーネルが”System is deadlocked on memory”というエラーを出力し，起動できなかった．これより，BitVisor と Unikraft を用いると 4 分の 1 のメモリ量で済むことがわかった．

SEV-tracker を用いずにクラウド VM 内で直接，クラウドサービスを実行した場合，Unikraft を用いるとメモリ割り当てを 48MB まで減らすことができた．これより，BitVisor を用いると 80MB 多くのメモリが必要になることがわかった．一方，Linux 単体では 412MB 以上ないと起動しなかったため，BitVisor と Unikraft を用いた場合の方がむしろ少ないメモリで済むことがわかった．

5.3 クラウドサービスの性能

SEV-tracker を用いた場合のクラウドサービスの性能を測定する実験を行った．Unikraft 上でベンチマークを動作させることができていないため，クラウド VM 内で BitVisor と Linux を動作させた．その結果をクラウド VM 内で KVM と Linux を動作させた場合と比較した．まず，Linux 上で sysbench を実行して CPU とメモリアクセスの性能を測定した．その結果を図 7 と図 8 に示す．この結果より，CPU 性能はほぼ同じであるが，メモリアクセス性能は BitVisor を用いた場合の方が 2 倍高くなることがわかった．

次に，Linux 上で Apache ウェブサーバを動作させ，同じネットワーク内の別のホストで ApacheBench を用いてリクエストを送信した．この実験では，BitVisor において通信の捕捉は行わなかった．得られたウェブサーバのスループットを図 9 に示す． この実験の結果，BitVisor を用いた場合に性能が 40%向上することが確認できた．

5.4 通信履歴の取得性能

まず，ユーザが BitVisor から通信履歴を取得できることを確認する実験を行った．現在の実装では，BitVisor 上で動作する Unikraft のネットワーク機能が不安定であるため，BitVisor 上で Linux を動作させた．この実験の結果，Linux が行った通信の情報をユーザが取得できることが確認できた．

次に，BitVisor から通信履歴を取得するのにかかる時間を測定した．この実験ではあらかじめ，BitVisor 内に 50 個の通信履歴を格納しておいた．そして，1～5 台のクラウド VM を用いて再帰的に通信履歴を取得する時間を測定した．i 番目の BitVisor は i-1 番目の BitVisor から要求を受け取ると i+1 番目の BitVisor に要求を送り，その BitVisor から応答を受け取ると i-1 番目の BitVisor に応答を返すようにした．通信履歴の取得時間を図 10 に示す．再帰的に取得することによってクラウド VM の台数に応じた時間がかかっていることがわかる．2 台から 3 台になった際に急激に取得時間が長くなっている原因については調査中である．

最後に，BitVisor において通信を捕捉するオーバヘッ

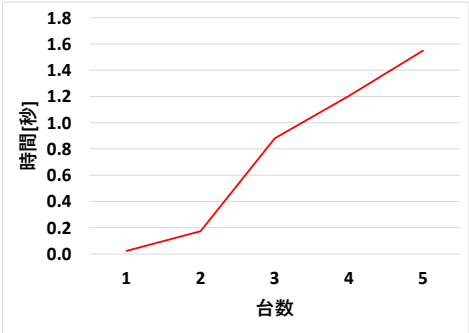

図 10 通信履歴の取得時間

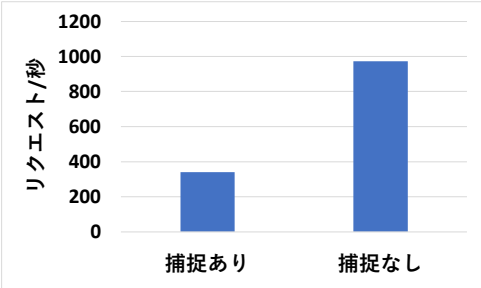

図 11 通信を捕捉するオーバヘッド

ドを調べるために，5.3 節と同様にウェブサーバのスループットを測定した．その結果は図 11 のようになり，通信を捕捉する場合にはスループットが 65%低下することがわかった．

6. 関連研究

Ryoan [11] は Intel SGX のエンクレイヴ内に NaCl[12] を用いてサンドボックスを作成し，その中でクラウドサービスを実行する．Ryoan はクラウドサービスの通信トポロジを強制することができる．クラウドサービスが別のサービスに機密情報を送信する際にはサービスに対応するラベルを追加し，応答を受け取るとラベルを削除する．ラベルに応じてサンドボックスがアクセスを制限することにより，機密情報の漏洩を防ぐ．また，ユーザが通信履歴を取得することも可能である．本研究では SEV と VM を用いてサンドボックスを作成する点が異なる．

LibSEAL [13] は SGX のエンクレイヴ内でクラウドサービスへの要求・応答のログを記録し，サービスの整合性違反を検出する．SGX によりログの整合性が担保されるため，サービスプロバイダは否認することができない．エンクレイヴ内で TLS の機能を提供することにより，既存のサービスに対しても透過的に利用させることができる．

CloudVisor [14] はネストした仮想化を用いて仮想化システムの下にセキュリティモニタを導入し，クラウド管理者からユーザの VM を保護する．ユーザ VM のメモリを保護するためにクラウド管理者からのアクセスを制限し，アクセスが必要な場合には暗号化や整合性検査を行う．また，ユーザ VM の仮想ディスクを保護するためにディスク I/O を監視し，書き込み時に暗号化して読み出し時には復号する．

SEVmonitor [15] は SEV を用いてメモリが暗号化された VM 内でエージェントを安全に動作させ，SEV で保護された別の VM に IDS をオフロード可能にしている．いくつかの手法が提案されているが，ネストした仮想化を用いる場合には監視対象 VM で BitVisor を動作させ，その上の VM で監視対象システムを動作させる．BitVisor 内でエージェントを安全に動作させ，IDS がエージェント経由でメモリデータを取得することで VM の監視を行う．SEVmonitor ではクラウドが BitVisor を用いてユーザの VM を監視することを想定しているが，SEV-tracker ではユーザの BitVisor を用いてクラウドの VM を監視する．

Xen-Blanket [16] はネストした仮想化を用いてクラウドが提供する VM 内でユーザのハイパーバイザを動作させることにより，クラウド間で VM マイグレーションを可能にする．通信を高速化するために，ユーザの管理 VM の OS 内で Blanket ドライバを動作させ，ユーザ・ハイパーバイザ経由でクラウドの管理 VM と通信を行う．クラウドの仮想化システムに変更が不要であるため，既存のクラウド上で実行することができる．

FlexCapsule [17] はクラウドサービスを動作させるアプリケーション VM をクラウド VM 内に作成する．アプリケーション VM をクラウド VM 間でマイグレーションすることにより，柔軟にサービスを統合したり，クラウド VM のスケールアップ・ダウンを行ったりすることができる．アプリケーション VM 内では OSv [18] や MiniOS などのライブラリ OS が提供され，ネストした仮想化のオーバヘッドが削減されている．

7. まとめ

本稿では，AMD SEV を用いてユーザがクラウド内のデータ流を安全に追跡・制御することを可能にするシステム SEV-tracker を提案した．SEV-tracker は SEV を用いてメモリを保護したクラウド VM にユーザ・ハイパーバイザを送り込むことで，クラウドサービスの通信を安全に追跡・制御する．ネストした仮想化のオーバヘッドを減らすために，ユーザ・ハイパーバイザおよびユーザ VM 内のゲスト OS として軽量な BitVisor と Unikraft をそれぞれ用いた．BitVisor 内にクラウドサービスの通信履歴を管理する機能を実装し，Unikraft を UEFI および BitVisor に対応させた．実験により，標準的な KVM と Linux を用いる場合に比べてクラウドサービスの性能を向上させられることがわかった．また，ユーザがユーザ・ハイパーバイザから短時間で通信履歴を取得できることを確認した．

今後の課題は，様々な Unikraft アプリケーションを動作させてデータ流の追跡を行えるようにすることである．まずは，Unikraft 上で通信を行って通信履歴を正しく取得で

きることを確認する．また，データ流の追跡だけでなく制御も行えるようにする予定である．さらに，SEV を用いて BitVisor を起動できるようにし，BitVisor 上で SEV を用いて Unikraft を起動できるようにすることも必要である．

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] Advanced Micro Devices, Inc. Secure Encrypted Virtualization API Version 0.24, 2020.
[2] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour. The Turtles Project: Design and Implementation of Nested Virtualization.
[3] T. Shinagawa, S. Hasegawa, T. Horie, Y. Oyama, S. Chiba, H. Eiraku, K. Tanimoto, M. Hirano, E. Kawai, Y. Shinjo, K. Omote, K. Kourai, K. Kono, and K. Kato. A Thin Hypervisor for Enforcing I/O Ddevice Security. In *Proceedings of International Conference on Virtual Execution Environments*, pp. 121–130, 2009.
[4] PwC. US Cybercrime: Rising Risk, Reduced Readliness, 2014.
[5] CyberArk Software. Global IT Security Service, 2009.
[6] S. Kuenzer, V. Badoiu, H. Lefeuvre, S. Santhanam, A. Jung, G. Gain, C. Soldani, C. Lupu, S. Teodorescu, C. Raducanu, C. Banu, L. Mathy, R. Deaconescu, C. Raiciu, and F. Huici. Unikraft: Fast, Specialized Unikernels the Easy Way. In *Proceedings of the 16th European Conference on Computer Systems*, pp. 376–394, 2021.
[7] EDK II Project. EDK II. https://github.com/tianocore/edk2.
[8] A. Dunkels. A Lightweight TCP/IP Stack. http://savannah.nongnu.org/projects/lwip/.
[9] 海洋軟件. UEFI 読本 ごった煮編, 2019.
[10] S. Christopherson. [PATCH] KVM: SVM: Treat SVM as unsupported when running as an SEV geust. https://www.spinics.net/lists/kvm/msg234582.html, 2021.
[11] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel. A Distributed Sandbox for Untrusted Computation on Secret Data. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*, 2016.
[12] Google, Inc. Native Client. https://developer.chrome.com/docs/native-client/, 2016.
[13] P. Aublin, F. Kelbert, D. O'Keeffe, D. Muthukumaran, C. Priebe, J. Lind, R. Krahn, C. Fetzer, D. Eyers, and P. Pietzuch. LibSEAL: Revealing Service Integrity Violations Using Trusted Execution. In *Proceedings of the 13th European Conference on Computer Systems*, 2018.
[14] F. Zhang, J. Chen, H. Chen, and B. Zang. CloudVisor: Retrofitting Protection of Virtual Machines in Multitenant Cloud with Nested Virtualization. In *Proceedings of the 23rd Symposium on Operating Systems Principles*, pp. 203–216, 2011.
[15] 能野, 光来. AMD SEV を用いてメモリが暗号化された VM に対する IDS オフロード. In *SwoPP 2021*, 2021.
[16] D. Wiliams, H. Jamjoom, and H. Weatherspoon. The Xen-Blanket: Virtualize Once, Run Everywhere. In *Proceedings of the 7th European Conference on Computer Systems*, pp. 113–126, 2012.
[17] K. Kourai and K. Sannomiya. Flexible Service Consolidation with Nested Virtualization and Library Operating Systems. *Software: Practice and Experience*, Vol. 50, No. 1, pp. 3–21, 2020.
[18] A. Kivity, D. Laor, G. Costa, P. Enberg, N. Har'El, D. Marti, and V. Zolotarov. OSv – Optimizing the Operating System for Virtual Machines. In *2014 USENIX Annual Technical Conference*, 2014.