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

安東 尚哉[1]　瀧口 和樹[1]　光来 健一[1]

概要：近年，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．その原因の一つとしてクラウドのサービスが複雑化し，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ことが挙げられる．一般的に，クラウドサービスが扱うデータの流れはユーザに公開されていないため，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には，ユーザが自分のデータを追跡・制御するためのプライバシ制御機構がクラウド内に必要となるが，ユーザはクラウドの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 を用いてネストした仮想マシン（VM）を保護することにより安全に通信の追跡・制御を行うシステム 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 VM 内でユーザの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情報の追跡・制御を行う．クラウド内でユーザ・ハイパーバイザを安全に実行するために，SEV を用いてユーザ・ハイパーバイザとクラウドを相互に保護する．SEV-tracker を BitVisor と Unikraft を用いて実装し，通信履歴が視覚的に表示できることおよび，クラウドサービスの性能について調べた．

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，クラウドのサービスが年々複雑化し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や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．

一般的に，クラウドサービスが扱うデータの流れはユーザに公開されていないため，ユーザはクラウドに送信したパーソナルデータがどこに転送されているのかを特定することはできない．クラウドはデータの流れを追跡したり制御したりするサービスや API も提供していないため，ユーザがデータの流れを追跡・制御することはできない．この問題を解決するには，ユーザが自分のデータを追跡・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．しかし，クラウドが提供する機構をユーザは完全には信頼することができず，正しくデータを追跡・制御できていることが保証できない．

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 はユーザのハイパーバイザをクラウド VM 内で実行し，プライバシ制御機構を動作させる．ユーザ・ハイパーバイザ上にユーザ VM を作成し，その中でユーザ用のクラウドサービスを動作させる．SEV を用いてクラウド VM とユーザ VM のメモリをそれぞれ暗号化することで，クラウドが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防ぐ．ユーザ・ハイパーバイザはユーザ VM のすべての通信を捕捉して解析し，ユーザが通信履歴を取得してデータ流を把握するために視覚的に表示できるようにする．

ネストした仮想化による必要リソースやオーバヘッドの増加を抑えるために，軽量なハイパーバイザおよびライブラリ 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 BitVisor[2] を用い，ユーザ VM による通信を追跡・制御できるようにした．BitVisor は通信情報をユーザ用のログサーバに送信し，ログサーバに蓄積された通信履歴をユーザが取得する．また，ユーザ VM 内でクラウドサービスを動作させる OS として，Unikraft[3] を用いた．Unikraft はクラウドサービ

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

スが必要とする最小限の OS の機能をアプリケーションにリンクする．SEV-tracker を用いて実験を行い，通信履歴が視覚的にわかりやすく表示できることを確認した．また，クラウドサービスのデプロイ時間や性能などについて調べた．

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 AMD 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 章では通信履歴の可視化や SEV-tracker の性能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増加傾向にあり，それに伴ってクラウドにおいて大量のパーソナルデータが扱われるようになっている．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．

その原因の一つとして，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複雑なサービスを提供するために，マイクロサービスなどのように複数のサービスを連携させること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．

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ユーザに公開されていないため，ユーザが利用したサービスに送信したパーソナルデータが他のどのサービスに送信されているのか，世界各地にあるデータセンタのどこにあるかなどのを特定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を制限することで情報漏洩を未然に防ぐこともできる．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が，それはプライバシ保護とのトレードオフである．

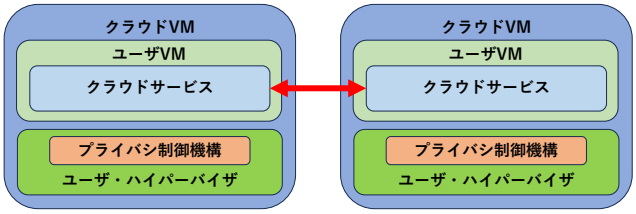


図 1: SEV-tracker のシステム構成

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頼することができず，パーソナルデータが正しく追跡・制御されていることを保証できない．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，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，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 のシステム構成は図 1 のようになる．SEV-tracker では，ユーザが自身のハイパーバイザをクラウドに送り込み，ネストした仮想化 [6] を用いてクラウドの VM 内で実行する．ユーザ・ハイパーバイザをクラウド VM 内で実行するのは，クラウドがインフラとして実行しているハイパーバイザを置き換えるのは現実的ではないためである．ユーザ・ハイパーバイザ上にユーザ VM を作成し，ユーザ VM 内でそのユーザ用の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すべての通信をユーザ・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．

一般的に，クラウドとユーザは互いに信頼できないため，クラウド内でユーザ・ハイパーバイザをどのようにして安全に実行するかが問題となる．クラウドはクラウド VM 上で動作するユーザ・ハイパーバイザよりも高い権限を持っているため，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，通信情報が改竄されたりして，正常にデータの追跡・制御が行えなくなる恐れがある．さらに，ユーザ・ハイパーバイザはユーザ VM 上で動作するクラウドサービスよりも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もできる．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 AMD SEV[1] を用いることでユーザ・ハイパーバイザとクラウドの間での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．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，ユーザ・ハイパーバイザが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 Nested SEV[7] で保護することにより，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，サービスの実行により多くのリソースを必要とし，クラウドサービスの性能も低下する．この影響を抑えるために，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，ハイパーバイザが必要とするリソースを減らし，性能低下を防ぐことができる．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を追跡・制御する必要のないデバイスは仮想化する必要がない．デバイス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，仮想デバイスの性能を改善することができる．

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 を提供する．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高速化することもできる．SEV-tracker はユーザごとにクラウドサービスを起動するため，クラウドサービスが用いる OS の起動性能は重要である．

SEV-tracker はユーザ VM のすべての通信を監視することにより，データ流の追跡・制御を可能にする．ユーザ VM 内のクラウドサービスが通信を行った際に，ユーザ・ハイパーバイザがパケットを捕捉し，その中のプライバシ制御機構がパケットヘッダを解析する．そして，通信情報をユーザごとに用意したログサーバに送信する．ログサーバはクラウドの VM 上で起動しておき，SEV を用いてクラウドから保護する．ログサーバに送信した通信情報を保護するために，暗号化，整合性検査，シーケンス番号の付与を行う．ユーザはログサーバに問い合わせることによって通信履歴を取得し，視覚的に表示することでクラウドサービスのデータ流を把握する．ユーザ・ハイパーバイザはユーザのポリシーに沿わない通信が行われた場合にはパケットを破棄し，通信の制御も行う．

4. 実装

軽量なユーザ・ハイパーバイザとして BitVisor[2] を用い，ユーザ VM の軽量なゲスト OS として Unikraft[3] を用いて SEV-tracker を実装した．

4.1 クラウドサービスの高速起動

SEV-tracker はユーザが送り込んだ BitVisor をクラウドの VM 内で実行することによりプライバシ制御を行う．ユーザはディスクイメージを KVM で用いられる qcow2 形式で作成し，UEFI が認識できるように FAT32 形式でフォーマットする．ディスクイメージには BitVisior を UEFI で起動するために必要なブートローダと BitVisior 本体が含まれる．また，BitVisior とクラウドサービスを自動起動するために用いられる UEFI シェル用のスクリプトも含まれる．作成したディスクイメージはユーザがクラウドサービスを使う際にクラウドのサーバへ転送する．ユーザが利用するクラウドサービスが他のクラウドサービスにアクセスする場合には，クラウドがディスクイメージを再帰的に他のサーバへ転送する．

SEV-tracker のシステム構成を図 2 に示す．クラウドのインフラとして提供されるハイパーバイザとして，Nested SEV に対応した KVM を用いる．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，クラウド VM のファームウェアとしてクラウドが提供する OVMF[8] を用いる．クラウド VM はユーザが送り込んだディスクイメージとクラウドが用意するクラウドサービスのディスクイメージを使用する．クラウドサービスのディスクイメージには Unikraft を用いて作成された UEFI アプリケーションとそれを起動するための UEFI シェル用のスクリプトが含まれる．クラウドサービスはファイルシステムとして，クラウドのホストファイルシス

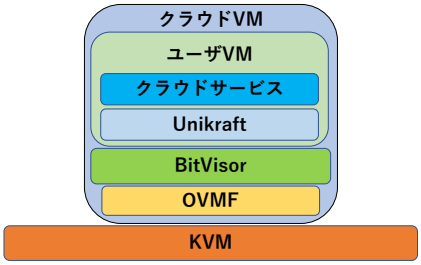


図 2: BitVisor と Unikraft を用いたシステム構成

テムを利用する．起動時にリモートアテステーションを用いることで，OVMF，ユーザ・ハイパーバイザ，クラウドサービスがそれぞれ正しく実行されていることを確認できる．

SEV-tracker はユーザの要求でクラウドサービスを起動する時に，KVM 上にクラウド VM を作成して OVMF を用いて UEFI シェルを実行する．UEFI シェルはユーザが送り込んだディスクイメージ内のスクリプト（startup.nsh）を実行することで，BitVisor を自動起動する．デフォルトの設定では UEFI シェルの起動後 5 秒経過しないとスクリプトが実行されないため，NVRAM にオプションを書き込むことで即座に実行されるようにした．このスクリプトはディスクイメージ内のブートローダを実行して BitVisior を起動する．BitVisior がユーザ VM を作成して起動すると，ユーザ VM の実行環境で UEFI シェルに制御が戻ってくる．そこで，startup.nsh はクラウドのディスクイメージ内のシェルスクリプトを呼び出すことによりクラウドサービスを起動する．

4.2 通信の追跡・制御

SEV-tracker では，BitVisor がユーザ VM のすべての通信を横取りしてパケットを解析し，通信の追跡・制御を行う．通信の捕捉には BitVisior の pass モジュールを用いる．pass モジュールはゲスト OS にネットワークデバイスを透過的に使用させつつ，ゲスト OS の通信をフックしてパケットをキャプチャすることを可能にする．BitVisior は捕捉したパケットのイーサネットヘッダ，IP ヘッダ，TCP もしくは UDP ヘッダを解析し，宛先と送信元アドレス，宛先と送信元ポートをそれぞれ取得する．取得した通信情報が以前取得したものと一致する場合は何もせずにパケットの転送処理を継続する．この際に，ユーザが事前に決めたポリシーに従ってパケットの破棄を行う．

BitVisor が取得した通信情報は UDP を用いる syslog プロトコルでユーザのログサーバへ送信される．ログサーバはユーザがクラウド内で VM を用いて起動しておく．BitVisor は事前に設定したログサーバの IP アドレスとゲートウェイまたはログサーバの MAC アドレスから UDP パケットを作成してログの送信を行う．この際に，BitVisior に内包されている lwip[9] が提供するネットワーク機能は使用しない．ログサーバへ送信する通信情報にはクラウドによるログの破棄を検知するためのシーケンス番号を付与する．また，ログサーバへ送信した通信情報がクラウドに盗聴されるのを防ぐため，AES を用いた暗号化を行う．さらに，通信情報がクラウドによって改竄されたことを検知できるように，通信情報のメッセージ認証コード（MAC）を計算してログサーバへ送信する．

暗号化された通信情報を受けとったログサーバは通信情報を復号し，シーケンス番号をチェックする．番号が連続していない通信情報を受け取った場合には，途中の通信情報が破棄された可能性があると記録する．ただし，UDP 通信では一部のパケットが届かないことや順番が入れ替わることもあるため，クラウドによって破棄されたと断定はできない．次に，受信した通信情報から MAC を計算し，受信した MAC の値と比較することにより整合性チェックを行う．MAC の値が一致しない場合には通信情報が改竄された可能性があると記録する．通信情報の整合性が確認できた場合は，ログサーバのデータベースに重複なく保存する．

4.3 通信履歴の可視化

SEV-tracker はクラウドサービスの通信履歴を可視化することで，クラウド内のデータ流をユーザに視覚的にわかりやすく表示する．通信履歴の可視化には net-glimpse[10] を用いる．net-glimpse は NIC からリアルタイムにパケットを取得して，通信の状況をブラウザ上で視覚的に表示する．ホストの IP アドレスがノードとして表示され，ノード間でパケットの送受信が行われると矢印でその方向が示される．既に表示されているノードでパケットの送受信があるとノードは明滅する．矢印には通信で用いられているプロトコル名またはポート番号が表示される．

通信履歴はパケットの一部の情報のみを記録したものであるため，そのままでは net-glimpse を用いることはできない．そこで，ネットワークデバイスの代わりとなる TAP デバイスを作成する．TAP デバイスはイーサネットデバイスをシミュレートし，データリンク層を操作することができるデバイスである．通信履歴に含まれる IP アドレスとポート番号，プロトコル情報からパケットヘッダを再構成し，イーサネットフレームを生成する．MAC アドレスなど，通信履歴に記録されていないフィールドには適当な値を用いる．このイーサネットフレームを TAP デバイスに書き込むと，net-glimpse がそのパケットをキャプチャして可視化を行うことができる．

クラウドサービスの利用後に通信履歴の可視化を行う場合，一括ですべての情報を表示することでクラウドサービスが行った通信の全体像を把握することができる．一方，取得した通信履歴を少しずつ表示することで，クラウドサービスによる通信の流れを把握することができる．また，

クラウドサービスの利用中に定期的に通信履歴を取得して可視化を行うことで，ほぼリアルタイムにクラウドサービスのデータ流を把握することもできる．

4.4 Unikraft のネットワーク対応

BitVisior は Unikraft の通信を捕捉するために準パススルードライバを用いる．BitVisior 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，Unikraft のネットワークドライバはこのような書き込みを行わないため，virtio-net が利用できるようにならない．そこで，BitVisior が 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ようにした．

BitVisor は Unikraft の通信を捕捉するために NIC の割り込みを利用する．しかし，Unikraft 0.15 は UEFI で起動すると割り込みが送られてこないため，virtio-net が利用できなかった．ポーリングでパケットの受信を行うようにすれば virtio-net が利用できるようになるが，BitVisor は通信を捕捉できない．この問題は OVMF を少し古いバージョンに変更することで回避することができた．さらに調査を行った結果，新しいバージョンの OVMF は PIC の割り込みマスクレジスタのビットをセットして，すべての IRQ をマスクしてしまうことが原因であることが分かった．Unikraft が IRQ 2 番のマスクをクリアするようにすることでこの問題を解決した．

4.5 BitVisor の Nested SEV 対応

Nested SEV [7] の方式の一つである SEV 仮想化を用いて，KVM 上に作成されたクラウド VM の中で BitVisor を動作させられるようにした．そのために，QEMU-KVM が提供する仮想 AMD セキュアプロセッサ（AMD-SP）用のデバイスドライバを BitVisor に実装した．仮想 AMD-SP は SEV の管理用コマンドを提供する AMD-SP を仮想化したものである．BitVisor が仮想 AMD-SP のコマンドを実行する際に渡すデータは仮想 AMD-SP がアクセスできるように暗号化を解除する．また，BitVisor は基本的にデバイスをパススルーするため，仮想 AMD-SP がユーザ VM に見えてしまわないようにするために隠蔽する．

クラウド VM 内で UEFI を使って起動した BitVisor はユーザ VM を作成し，その VM 内で同じ UEFI を使って Unikraft を起動する．しかし，クラウド VM とユーザ VM は異なる暗号鍵を用いるため，ユーザ VM 内では UEFI の実行を継続できない．そこで，ユーザ VM 用のブートローダを実装して Unikraft を起動できるようにした．BitVisor はユーザ VM にロードした Unikraft を仮想 AMD-SP のコマンドを実行してユーザ VM の鍵を用いて暗号化する．

表 1: ホストマシンの構成

CPU	AMD EPYC 7402P	AMD EPYC 7262P
メモリ	256GB	128GB
NIC	10GbE	
OS	Linux 5.4	
ハイパーバイザ	QEMU-KVM 4.2.1	

5. 実験

SEV-tracker を用いてログサーバから取得した通信履歴が可視化できることを確認する実験を行った．この実験のために Unikraft 上で flask と python を用いて動作するウェブアプリケーションを作成した．また，クラウドサービスのデプロイ時間，必要な最小メモリ，通信情報を取得するオーバヘッド，サービスの性能を調べた．Nested SEV を用いた BitVisor と Unikraft の実行がまだ安定していないため，本実験は SEV を適用せずに行った．

クラウド内のホストとして，表 1 の 2 台のサーバを用いた．クラウド VM には仮想 CPU を 1 つ，メモリを 400MB 割り当て，ユーザ・ハイパーバイザとして BitVisior，ユーザ VM の OS として Unikraft0.15 を動作させた．比較として，ユーザ・ハイパーバイザとして KVM，ユーザ VM の OS として Linux 5.4 を動作させた．この場合には，クラウド VM にメモリを 4GB 割り当て，ユーザ VM にメモリを 400MB 割り当てた．クライアントとして，Intel Core i7-10700 の CPU，64GB のメモリ，1GbE の NIC を搭載したマシンを用いた．

5.1 クラウドサービスの通信履歴の可視化

SEV-tracker を用いて通信履歴を取得して可視化する実験を行った．この実験では 2 つのクラウドサービスを実行し，ユーザがサービス 1 と HTTPS 通信を行うと，サービス 1 は DNS サーバと通信して名前解決を行ってからサービス 2 と HTTPS 通信を行うようにした．さらに，サービス 2 は監視対象外の外部サービスと HTTPS 通信を行うようにした．この実験の結果，可視化された通信履歴を図 3 に示す．実際に行った通信と可視化の結果が一致しており，ユーザにクラウド内でのデータ流を視覚的にわかりやすく示せることが確認できた．

5.2 クラウドサービスのデプロイ時間

SEV-tracker を用いてユーザごとのクラウドサービスをデプロイするのにかかる時間を測定した．この実験では，ユーザがクラウドにディスクイメージを転送してクラウド VM を作成し，その中でユーザ・ハイパーバイザを起動してユーザ VM を作成し，クラウドサービスを起動してネットワークが使用できるようになるまでの時間をデプロイ時

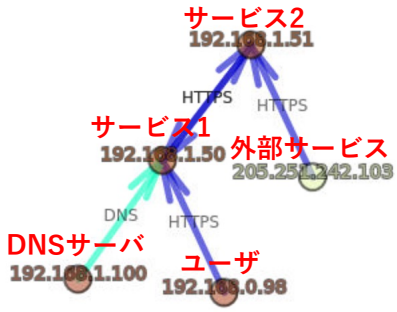


図 3: 可視化した通信履歴

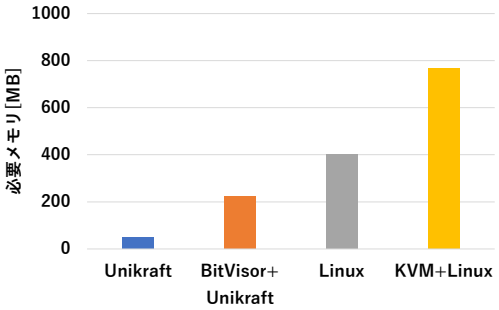


図 5: クラウドサービスの起動に必要な最小メモリ

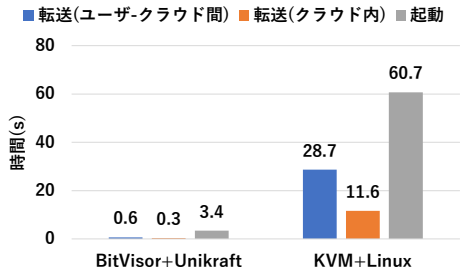


図 4: クラウドサービスのデプロイ時間の内訳

間とした．クラウド内で複数のサービスを用いる場合は，クラウド内でディスクイメージを転送する時間も含まれる．デプロイ時間の内訳として，ディスクイメージを転送する時間と BitVisor 上で Unikraft を用いたクラウドサービスを起動する時間を図 4 に示す．比較として，KVM を含むディスクイメージを転送する時間と KVM 上で Linux を起動する時間も測定した．

起動するクラウドサービスが 1 つの場合，SEV-tracker におけるデプロイ時間は 4 秒となった．その内訳は，ディスクイメージの転送が 0.6 秒，クラウドサービスの起動が 3.4 秒であった．SEV-tracker は KVM と Linux を用いた場合に比べて，49 倍高速にディスクイメージを転送し，22 倍高速にクラウドサービスを起動することができた．KVM をユーザ・ハイパーバイザとして用いるとディスクイメージが 328 倍になることが転送時間の増加の原因である．また，KVM を用いるにはクラウド VM 内で Linux を起動する必要があるため，起動時間が増加した．起動するクラウドサービスが 2 つになった場合でも，SEV-tracker を用いた場合のデプロイ時間の増加は 0.3 秒であった．これはユーザとクラウド間よりクラウド内のネットワークの方が高速であるためである．

5.3 クラウドサービスに必要な最小メモリ

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メモリ量を調べた．この実験では，クラウド VM とユーザ 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正常に起動できなくなるメモリ量を探索した．実験結果を図 5 に示す．SEV-tracker を用いて BitVisor と Unikraft を起動した場合，224MB までは正常に起動したが，それ以下に減らすと BitVisor がメモリ割り当てに失敗して起動できなくなった．一方，クラウド VM 内で KVM と Linux を起動した場合，704MB 以下のメモリを割り当てると Linux カーネルのメモリが不足して起動できなくなることがあった．これより，SEV-tracker は KVM と Linux を用いる場合の 29%のメモリ量で起動できることがわかった．

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．それ以下に減らすと，UEFI シェルが起動できなくなった．これより，BitVisor を用いると 176MB 多くのメモリが必要になることがわかった．また，Linux 単体では 400MB 以上のメモリが必要であるため，SEV-tracker の方が必要なメモリ量を抑えられることがわかった．

5.4 通信情報を取得するオーバヘッド

SEV-tracker において BitVisor がユーザ VM のパケットをキャプチャして通信情報をログサーバに送信するオーバヘッドを調べた．クラウドサービスとして固定のレスポンスを返すシンプルな HTTP サーバを動作させ，Apache Bench を用いて 1 人のユーザがリクエストを送った際のスループットを測定した．比較として，BitVisor がパケットをキャプチャしない場合（passthrough），パケットのキャプチャのみを行う場合（pass），ログサーバへの転送に BitVisor 内の lwIP を使う場合（ippass）についても調べた．

実験結果を図 6 に示す．パケットをキャプチャしない場合に比べて，通信情報を送信する場合には 10%のオーバヘッドがかかることがわかった．その内訳は，パケットをキャプチャすることによって 5%，通信情報を送信することによって 5%であった．これより，SEV-tracker がクラウドサービスの通信情報を取得するオーバヘッドはそれほど大きくないことがわかった．一方，lwIP を用いて通信情報の送信を行うとさらに 5%性能が低下することがわかった．

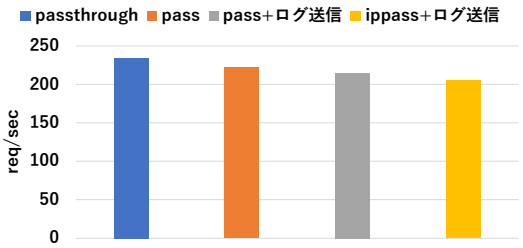
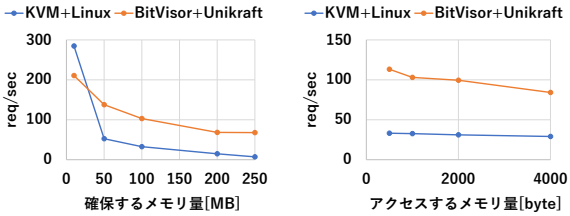


図 6: 通信情報を取得するオーバヘッド



(a) アクセスするメモリ量を固定 (b) 確保するメモリ量を固定

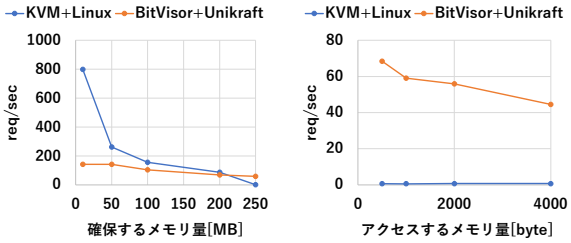
図 7: 処理毎にメモリを確保する場合の性能

5.5 クラウドサービスの性能

SEV-tracker を用いて動作させたクラウドサービスの性能を調べた．この実験ではまず，大量のメモリを使用して処理を行うサービスを模したクラウドサービスを用いた．このクラウドサービスは HTTP リクエストを受け取ると一定量のメモリを確保してアクセスを行った後，メモリを解放する．クラウドサービスがアクセスするメモリ量を 4KB のページあたり 1000 バイトに固定して，確保するメモリ量を変化させた時の性能を図 7a に示す．クラウドサービスが確保するメモリ量が増えると，KVM と Linux を用いる場合に比べて SEV-tracker の方が性能がよくなることがわかった．これは，同じ量のメモリをユーザ VM に割り当てた場合，Linux は Unikraft に比べて使用できるメモリが少ないため，メモリスワップが発生しやすくなることが原因である．また，Linux はデマンドページングを行うため，新しいページにアクセスするたびにページフォールトが発生することも原因である．

また，クラウドサービスが確保するメモリ量を 100MB に固定し，ページごとにアクセスするメモリ量を変化させた時の性能を図 7b に示す．SEV-tracker のほうが常に Linux より性能がよいが，アクセスするメモリ量が増えると性能が低下する割合が大きいことがわかった．これは Linux ではページフォールトとメモリスワップがボトルネックになっているため，メモリアクセス量の影響が相対的に小さいからだと考えられる．

次に，データをメモリ上に格納するインメモリデータベースを模したクラウドサービスを用いた．このクラウドサービスは起動時にメモリを確保し，リクエストを受け取るとそのメモリにアクセスして処理を行う．アクセスするメモリ量を 1000 バイトに固定し，起動時に確保するメモリ量を変化させた時の性能を図 8a に示す．KVM と Linux を用いた場合と比較して，確保するメモリ量が少ない場合には SEV-tracker の方が性能が低いことがわかった．これは Linux のデマンドページングの影響がなくなったためである．確保するメモリ量が増えるにつれて Linux はメモリスワップの影響を大きく受けるようになった．確保するメモリ量を 250MB に固定し，アクセスするメモリ量を変化させた時の性能を図 8b に示す．SEV-tracker を用いた場合，ページごとにアクセスするメモリ量が増えると性能が少しずつ低下することがわかった．



(a) アクセスするメモリ量を固定 (b) 確保するメモリ量を固定

図 8: 起動時にメモリを確保する場合の性能

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ラベルをつけ，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．同様に，AccTEE [13] は SGX のエンクレイヴ内で WebAssembly [14] を用いてサンドボックスを構築し，サンドボックス外部でリソース利用情報を安全に記録することができる．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．

CloudVisor [15] 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 [16] 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 [17] 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 [18] 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 [19] 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 を用いてネストした VM を保護することによって安全に通信の追跡・制御を行うシステム SEV-tracker を提案した．SEV-tracker はネストした仮想化を用いてクラウドの VM 内でユーザの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の追跡・制御を行う．クラウド内でユーザ・ハイパーバイザとクラウドサービスを安全に実行するために，SEV を用いてユーザ・ハイパーバイザとクラウドサービスを相互に保護する．実験により，クラウドサービスの通信履歴を可視化できることがわかった．また，クラウドサービスのデプロイ時間が非常に短いことが分かった．

今後の課題は，クラウド VM とユーザ VM に SEV を適用した状態で性能を測定することである．また，ユーザによりわかりやすく通信履歴の可視化を行うことも必要である．

謝辞

本研究の一部は，JST，CREST，JPMJCR21M4 の支援を受けたものである．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] Advanced Micro Devices, Inc. Secure Encrypted Virtualization API Version 0.24, 2020.
[2] 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.
[3] 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.
[4] PwC. US Cybercrime: Rising Risk, Reduced Readliness, 2014.
[5] CyberArk Software. Global IT Security Service, 2009.
[6] 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.
[7] 瀧口, 光來. Nested SEV: ネストした仮想化への AMD SEV の適用. コンピュータシステム・シンポジウム, 2022.
[8] EDK II Project. EDK II. https://github.com/tianocore/edk2.
[9] A. Dunkels. A Lightweight TCP/IP Stack. http://savannah.nongnu.org/projects/lwip/.
[10] krisitan lange. *net-glimpse*. https://github.com/kristian-lange/net-glimpse.
[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] D. Goltzsche, M. Nieke, T. Knauth, and R. Kapitza. AccTEE: A WebAssembly-based Two-way Sandbox for Trusted Resource Accounting. In *Proceedings of the 20th International Middleware Conference*, 2019.
[14] A. Haas, A. Rossberg, D. Schuff, B. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien. Bringing the Web Up to Speed with WebAssembly. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2017.
[15] F. Zhang, J. Chen, H. Chen, and B. Zang. CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization. In *Proceedings of the 23rd Symposium on Operating Systems Principles*, pp. 203–216, 2011.
[16] 能野, 光來. 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.
[17] 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.
[18] 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.
[19] 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.