

機密 VM 内の隔離環境でのカーネルメモリ解析による 通信の安全かつ軽量の追跡

新矢 将宗^{1,a)} 光来 健一¹

概要：近年、多様なサービスがクラウド上で稼働し、機密データもクラウド上に集約されるようになっていく。さらに、マイクロサービス化によりサービス間通信が増加しており、データフローが複雑化している。そこで、機密仮想マシン (機密 VM) 内で VM を用いてクラウドサービスを動作させることで、ユーザが安全に通信を追跡可能にするシステムが提案されている。しかし、このシステムではネストした仮想化とパケットキャプチャのオーバーヘッドが大きい。本稿では、機密 VM 内に作成した隔離環境においてカーネルメモリを解析することにより、安全かつ軽量に通信を追跡する SockTrawl を提案する。機密 VM 内の隔離環境は Intel TDX の TD Partitioning および AMD SEV-SNP の VM Privilege Level (VMPL) を用いて作成し、その中で Secure VM Service Module (SVSM) を動作させる。SVSM において Linux カーネルの TCP ハッシュテーブルを定期的に走査し、TCP 通信の情報を取得する。Intel プロセッサと AMD プロセッサを搭載した 2 台のサーバを用いて、SockTrawl の追跡能力およびオーバーヘッドを調べた。

キーワード：機密 VM, VM イントロスペクション, TCP 通信追跡

Secure and Lightweight Communication Tracking with Kernel Introspection in Isolated Environments within Confidential VMs

MASAMUNE SHINYA^{1,a)} KENICHI KOURAI¹

Abstract: In recent years, clouds run various services and store sensitive data. In addition, microservices increase inter-service communication and complicate data flows. To enable users to securely track data flows in clouds, previous work runs a cloud service in a virtual machine (VM) within a confidential VM (CVM) and monitors the communication. However, it suffers from significant overhead caused by nested virtualization and packet capturing. This paper proposes SockTrawl for tracking communication securely and efficiently by introspecting kernel memory in an isolated environment within a CVM. SockTrawl creates this isolated environment using TD Partitioning in Intel TDX or VM Privilege Level (VMPL) in AMD SEV-SNP and runs the Secure VM Service Module (SVSM) in it. It periodically scans the Linux kernel's TCP hash table to obtain information on TCP connections. We examined the tracking capability and overhead of SockTrawl on two servers equipped with Intel and AMD processors.

Keywords: Confidential VM, VM Introspection, TCP Communication Tracking

1. はじめに

近年、多様な IT サービスがクラウド上で稼働し、顧客情報などの機密データもクラウド上に集約されるようになって

きている。一方で、クラウド上に保管されたデータは広範囲からアクセスされる可能性があり、実際にクラウドの不適切な設定や脆弱性を狙った攻撃が行われている。そのため、クラウド上における通信の流れをユーザが安全に追跡できる機構が求められている。しかし、このような機構をクラウド側が提供する場合、ユーザがクラウドを無条件に信頼することが前提となっており、クラウド内部からの情

¹ 九州工業大学
Kyushu Institute of Technology

^{a)} sny430@ksl.ci.kyutech.ac.jp

報漏洩が発生していないことを保証することはできない。

そこで、機密仮想マシン (VM) とネストした仮想化を用いてクラウド内の通信を安全に追跡する SEV-tracker [1] が提案されている。SEV-tracker では、クラウドから保護された機密 VM 内でユーザのハイパーバイザを実行し、その上に作成したユーザ VM 内で動作するクラウドサービスの通信を追跡する。SEV-tracker では軽量のユーザ・ハイパーバイザを用いているものの、機密 VM 内でユーザ VM を動作させるために必要となる Nested SEV [2] のオーバーヘッドが大きい。また、ユーザ VM が送受信するすべてのパケットをユーザ・ハイパーバイザで捕捉して解析するため、パケットキャプチャによるオーバーヘッドが発生する。

本稿では、ユーザの追跡機構をクラウド・ハイパーバイザおよびクラウドサービスから保護しつつ、クラウドサービスの通信を軽量に追跡する SockTrawl を提案する。SockTrawl は機密 VM 内に Nested SEV を用いるよりも軽量に隔離環境を作成し、その中でユーザの追跡機構を動作させる。Intel TDX を用いる機密 VM では TD Partitioning [3] を用い、機密 VM 内に作成した軽量 VM 内でクラウドサービスを動作させて追跡を行う。AMD SEV-SNP を用いる機密 VM では VM Privilege Level (VMPL) を用い、機密 VM の中のより低い特権レベルでクラウドサービスを動作させて追跡を行う。追跡機構を動作させるための基盤として、隔離環境の中で SVSM [4] を実行する。

SVSM の一機能として動作する追跡機構は、クラウドサービスが用いるゲスト OS のカーネルメモリを解析し、ゲスト OS の持つ TCP 接続情報を取得する。追跡機構は接続確立状態の TCP ソケットだけでなく、接続が切断された後の TIME_WAIT 状態のソケットも追跡することにより、TCP 接続情報の取得タイミングに依存せずにすべての TCP 接続の情報を取得する。SockTrawl は CPU のタイムスタンプカウンタを利用して追跡機構を定期的に実行する。これにより、クラウドサービスが送受信する全パケットをハイパーバイザにおいて捕捉することによるオーバーヘッドを削減し、通信の追跡に必要な TCP 接続情報のみを効率よく取得する。

SockTrawl を TDX の TD Partitioning を用いた SVSM [5], [6] と SEV-SNP の VMPL を用いた SVSM [4] に実装し、Intel プロセッサと AMD プロセッサを搭載した 2 台のサーバを用いて実験を行った。まず、追跡機構がクラウドサービスによる TCP 接続を追跡可能かどうかを確認した。その結果、いずれの環境でも TCP 接続を 100% 追跡できることを確認した。次に、機密 VM 内で隔離環境を用いるオーバーヘッドおよび追跡機能のオーバーヘッドを測定した。その結果、TDX では SVSM 自体の導入による性能低下が確認された。一方、SVSM のみを動作させた場合と比較して、通信の追跡によるオーバーヘッドは小さいことを確認した。

以下、2 章では本研究の背景について述べる。3 章では機密 VM 内の隔離環境においてカーネルメモリを解析して TCP 通信を追跡する SockTrawl を提案する。4 章では SockTrawl の実装について詳述し、5 章で SockTrawl の評価実験およびその結果について述べる。最後に、6 章で関連研究について触れ、7 章で本稿をまとめる。

2. 背景

2.1 クラウドサービスの安全な追跡

近年、クラウドでは、多数のサービスを連携させて複雑なサービスを提供することが一般的になっている。特に、マイクロサービスを用いて大規模化したサービスでは、サービス間の通信が増加し、そのデータフローが複雑化している。マイクロサービスでは各サービスごとにセキュリティ設定やアクセス制御が異なる場合が多く、設定漏れや誤設定に起因する脆弱性が生じるリスクが高い。その結果、機密データが様々なサービスに転送されることがクラウドからの情報漏洩を増加させる要因になっている。しかし、サービス間の通信やデータの流れが複雑化しているようなサービスにおいては、ユーザにとって自身のデータがどのサービスに転送されたのかを把握することは難しい。

ユーザがクラウドからの情報漏洩を把握するには、クラウドサービスによる通信を追跡し、データの流れを確認できるようにすることが有効である。しかし、クラウドによって提供される追跡機構を用いる場合、クラウドの管理者やクラウドに侵入した攻撃者によって追跡機構が無効化されたり、記録された通信情報を改ざんされたりするため、ユーザはその追跡結果を完全に信頼することはできない。また、クラウド内でユーザの追跡機構を動作させたとしても、同様に攻撃を受ける恐れがある。そのため、ユーザがクラウドを信頼することなく通信を追跡できる機構が必要である。

そこで、ユーザの追跡機構を用いてクラウドサービスの通信を安全に追跡することを可能にする SEV-tracker [1] が提案されている。SEV-tracker はユーザの追跡機構を機密 VM 内に配置することにより、クラウドのハイパーバイザを含むクラウド全体から保護する。機密 VM は、Intel TDX [7] や AMD SEV-SNP [8], Arm CCA [9] などの高信頼実行環境 (TEE) を用いて保護された VM である。これらの TEE はハードウェアの機能を用いて VM のメモリやレジスタを暗号化したり整合性をチェックしたりすることにより、信頼できないホスト OS やハイパーバイザからのアクセスを防ぎ、VM 内のデータを保護する。追跡機構がクラウドサービスの通信を追跡できるようにするために、クラウドサービスも同じ機密 VM 内に配置する。

SEV-tracker はさらに、ユーザの追跡機構をクラウドサービスからも隔離して保護する。そのために、Nested SEV [2] を用いて機密 VM 内でユーザ・ハイパーバイザを動作さ

せ、その上に作成されるユーザ VM 内でクラウドサービスを動作させる。Nested SEV は SEV-SNP で保護された機密 VM 内で VM を動作させることを可能にする機構である。ユーザの追跡機構をユーザ・ハイパーバイザ内で動作させることにより、ユーザ VM 内に隔離したクラウドサービスからの攻撃を防ぐ。追跡機構はユーザ VM 内のクラウドサービスが通信を行うたびにパケットを捕捉してヘッダを解析し、通信に関する情報を取得する。取得した情報は別の機密 VM 内で動作するログサーバへ送られ、ユーザはログサーバから通信情報を取得する。

しかし、SEV-tracker は Nested SEV を用いて機密 VM 内でユーザ VM を動作させるため、オーバーヘッドが大きい。ユーザ・ハイパーバイザとして軽量な BitVisor [10] を用いているが、ネストした仮想化を用いて VM 内で VM を動作させるだけで Web サーバの性能が 0~46% 低下する。機密 VM に Nested SEV を適用すると性能がさらに 4~43% 低下する [2]。また、ユーザ VM では汎用 OS の代わりにライブラリ OS の Unikraft [11] を用いることで VM の実行に必要なリソースを削減しているが、Unikraft を用いることによる性能低下も生じる。それに加えて、ユーザ VM の全パケットを捕捉するオーバーヘッドも大きい。SEV-tracker では BitVisor の pass モジュールを使うことにより、ユーザ VM が NIC に直接アクセスしつつアクセスを横取りできるようにしているが、それでも Web サーバの性能が 10% 低下する。

2.2 脅威モデル

本稿では、TEE を提供する CPU および、機密 VM 内で動作する追跡機構とその実行基盤は信頼できるものとし、脆弱性はないものと仮定する。一方、クラウド内のホスト OS やハイパーバイザ、それらと同等の権限を持つクラウド管理者や攻撃者、およびクラウドサービスは信頼しない。クラウド管理者や攻撃者および、クラウドサービスによる追跡機構への攻撃を対象とするが、ゲスト OS のカーネル内の TCP 接続情報を改ざんする攻撃は対象としない。

3. SockTrawl

本稿では、ユーザの追跡機構をクラウド・ハイパーバイザおよびクラウドサービスから保護しつつ、クラウドサービスの通信を軽量に追跡する SockTrawl を提案する。図 1 に SockTrawl のシステム構成を示す。SockTrawl は機密 VM 内に Nested SEV を用いるよりも軽量の隔離環境を作成し、その中でユーザの追跡機構を動作させる。隔離環境の外でクラウドサービスを動作させ、追跡機構はクラウドサービスが用いるゲスト OS のカーネルメモリを解析することによって通信情報を取得する。これにより、クラウドサービスの全パケットを捕捉するよりも小さなオーバーヘッドで通信を追跡する。

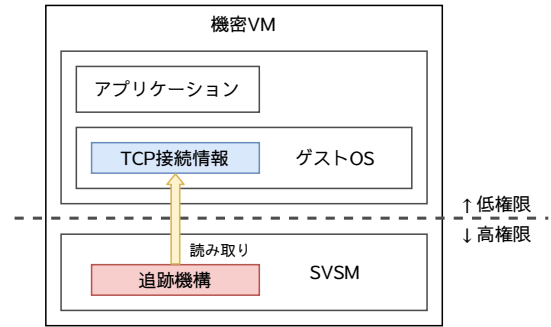


図 1: SockTrawl のシステム構成

SockTrawl は Intel TDX と AMD SEV-SNP が提供するそれぞれの機構を用いて、追跡機構を動作させるための軽量の隔離環境を作成する。TDX を用いる機密 VM では TD Partitioning [3] と呼ばれる機構を用いる。TD Partitioning は機密 VM 内を 4 つのパーティションに分割し、1 つの L1 VM と 3 つの L2 VM を構成することを可能にする。L1 VM は L2 VM のメモリを参照することができる。SockTrawl は L1 VM を隔離環境として用いてユーザ・ハイパーバイザおよび追跡機構を動作させ、L2 VM をユーザ VM として用いてクラウドサービスを動作させる。このシステム構成は Nested SEV と似ているが、TD Partitioning はネストした仮想化のオーバーヘッドを削減することができる。例えば、L2 VM には機密 VM のメモリが分割して割り当てられ、メモリは仮想化されない。また、L2 VM とユーザ・ハイパーバイザの間の遷移はクラウド・ハイパーバイザを経由せずに直接行うことができる。

一方、SEV-SNP を用いる機密 VM では VM Privilege Level (VMPL) と呼ばれる機構を用いる。VMPL は機密 VM 内に 4 つの特権レベルを設け、メモリに対するアクセス権限を特権レベルごとに設定することを可能にする。最も高い特権レベルの VMPL0 は機密 VM 内のすべてのメモリを参照することができる。SockTrawl では VMPL0 を隔離環境として用いて追跡機構を動作させ、より低い特権レベルの VMPL2 でクラウドサービスを動作させる。TDX とは異なり、VMPL は新たな特権レベルを導入するだけであるため、クラウドサービスをネストした仮想化のオーバーヘッドなしで実行することができる。

SockTrawl は機密 VM 内の隔離環境で Secure VM Service Module (SVSM) [4] を動作させ、その一機能として通信の追跡機能を提供する。SVSM は機密 VM 内でゲスト OS に対してセキュリティ機能やデバイス機能を提供する特権モジュールである。SVSM と追跡機能はメモリ安全な Rust 言語で実装され、機密 VM 内の高い権限で動作するコードをより安全に実行することができる。TDX を用いた機密 VM と SEV-SNP を用いた機密 VM とで隔離環境が大きく異なるが、SVSM が隔離環境の違いを吸収するた

め、ほぼ同じ追跡機構を用いることができる。具体的には、追跡機構を同様の方法で呼び出し、追跡機構が同様のインタフェースを用いてゲスト OS のメモリにアクセスすることができる。

SockTrawl は CPU のタイムスタンプカウンタを用いることで追跡機構を周期的に実行する。SVSM は一定時間ごとに処理を実行することができるタイマ機能を提供していないが、様々なイベントによって呼び出される。TDX を用いた機密 VM の場合、様々な要因で L2 VM からの VM Exit が発生して L1 VM で動作する SVSM が呼び出される。SEV-SNP を用いた機密 VM の場合、VM Exit は発生しないが、SVSM の機能を利用するためにゲスト OS から SVSM が明示的に呼び出される。そこで、SVSM が呼び出された際にタイムスタンプカウンタの値を取得して経過時間を計測し、前回の追跡機構の実行から一定時間が経過していれば追跡機構を呼び出す。この周期処理の頻度は任意の値に設定することが可能であり、頻度を高くするとよりリアルタイムに通信情報を取得でき、頻度を低くすると追跡のオーバーヘッドを減らすことができる。全パケットを捕捉する従来手法ではこのような調整を行うことはできない。

SVSM 内で動作する追跡機構は定期的にゲスト OS のカーネルメモリを解析することにより、ゲスト OS の持つ TCP 接続情報を取得する。そのために、ゲスト OS のメモリ上のページテーブルにアクセスして OS データの仮想アドレスを物理アドレスに変換する。TCP 接続情報は接続確立状態の TCP ソケット一覧から取得するが、この TCP ソケットは接続が切断されると解放される。短時間で切断される TCP 接続の追跡漏れを防ぐには、短い周期で TCP 接続情報を取得する必要があるが、取得頻度を増やすと監視によるオーバーヘッドが増加する。そこで、SockTrawl は TCP 接続の切断後も一定時間保持される TIME_WAIT 状態の TCP ソケットも追跡対象とする。これにより、接続の確立から切断までの時間が追跡周期より短い場合でも、TCP 接続情報を取得することができる。

4. 実装

SockTrawl を TDX の TD Partitioning を用いた SVSM (TDX-SVSM) [5], [6] と、SEV-SNP の VMPL を用いた SVSM (SEV-SVSM) [4] にそれぞれ実装した。

4.1 タイムスタンプカウンタを用いた周期処理

SockTrawl は CPU が提供するタイムスタンプカウンタ (TSC) を用いて経過時間を計測する。TSC は CPU が保持する 64 ビットのカウンタであり、RDTSC 命令を用いて現在値を取得することができる。SockTrawl は追跡周期を TSC クロック数へ変換するために、CPUID 命令を用いて TSC 周波数を取得する。TDX-SVSM では、L2 VM において VM Exit が発生した際に実行される VM Exit ハンド

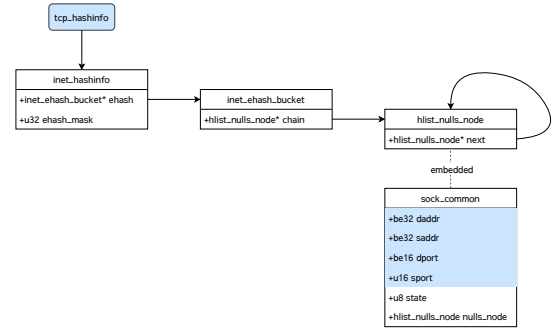


図 2: TCP 接続情報の取得に用いる Linux カーネル構造体

ラ内で分周比と基準クロックを取得し、そこから TSC 周波数を算出する。SEV-SVSM では、ゲスト OS から明示的に呼び出された時に実行される Guest Exit ハンドラ内で同様に TSC 周波数を算出する。そして、時間に TSC 周波数を乗じることで対応する TSC クロック数を得る。

SVSM 内の追跡機構は VM Exit または Guest Exit が発生するたびに TSC の値を取得し、取得した TSC の値と前回、追跡を実行した際の TSC の値との差分を調べる。差分値が設定した周期を超過している場合には、TCP 接続情報の取得を行う処理を呼び出す。複数の仮想 CPU から同時に SVSM へ遷移した場合に同一周期の処理が重複して実行されることを防ぐため、この処理はアトミックに実行する。これにより、追跡機構のために新たなタイマ割り込みなどを発生させることなく、既存の SVSM への遷移を利用して周期処理を実現する。ただし、設定した周期が経過しても追跡処理は直ちには実行されない。設定周期が経過した後に最初の VM Exit または Guest Exit が発生した時点で追跡処理を実行するため、実際の追跡間隔は設定周期より長くなる場合がある。

4.2 TCP 接続情報の取得

SVSM 内の追跡機構は、ゲスト OS として動作する Linux カーネルのメモリ上にある TCP ハッシュテーブルを取得し、そのすべてのバケットを順に走査する。この処理で参照する Linux カーネルの構造体の関係を図 2 に示す。TCP ハッシュテーブルである tcp_hashinfo は inet_hashinfo 構造体として定義されており、TCP 接続などを管理するハッシュテーブルの実体である inet_eshash_bucket 構造体へのポインタやバケット数を保持している。また、inet_eshash_bucket に含まれる各バケットは TCP ソケットを連結したリストへのポインタを保持している。このリストは hlist_nulls_node 構造体を用いて構成されており、SockTrawl は各ノードの next ポインタを順にたどる。hlist_nulls_node は TCP ソケットの共通情報を保持する sock_common 構造体内に埋め込まれているため、ノードの仮想アドレスから構造体内のオフセットを減じることで、対応する sock_common 構造体の先頭アドレスを求める。

SockTrawl は各 sock_common 構造体からアドレスと TCP の状態を取得し、IPv4 のソケットのうち TCP_ESTABLISHED 状態および TCP_TIME_WAIT 状態のものを追跡対象とする。対象となるソケットから送信元 IP アドレス、宛先 IP アドレス、送信元ポート番号、宛先ポート番号を読み出し、TCP 接続情報としてログに出力する。同じ TCP ソケットがハッシュテーブルを走査するたびに何度も取得される可能性があるため、ログへ出力したソケットのアドレスを SVSM 内に保持することで、同一の TCP 接続を重複して出力することを抑止する。この一連の処理はゲスト OS にのみ依存し、TEE や SVSM の実装に依存しないため、TDX-SVSM と SEV-SVSM の両方で同一の実装を用いることができる。

4.3 KASLR への対応

SockTrawl はゲスト OS が Kernel Address Space Layout Randomization (KASLR) を有効にしている場合には、それを考慮して TCP 接続情報の解析を行う。TCP ハッシュテーブルを取得するためには、Linux カーネル内の tcp_hashinfo シンボルの仮想アドレスを特定する必要がある。しかし、Linux では KASLR によってカーネルシンボルのアドレスが起動ごとに変化する。そこで、あらかじめゲスト OS 上で /proc/kallsyms を参照し、linux_banner シンボルの仮想アドレスから tcp_hashinfo シンボルの仮想アドレスへの相対オフセットを求めて SVSM に登録しておく。

SockTrawl は SVSM 起動時に、カーネルメモリ上の linux_banner の格納場所を探索する。linux_banner には Linux のバージョン等からなるバナーの ASCII 文字列が格納されているため、linux_banner が配置される可能性のあるメモリ領域を順に読み出し、バナーの文字列であるか否かを確認する。見つけた linux_banner の仮想アドレスに登録しておいた相対オフセットを加えることで、tcp_hashinfo の実行時の仮想アドレスを求める。KASLR によってカーネル全体の配置アドレスが変化しても、同一のカーネルイメージにおける二つのシンボル間の相対オフセットは変化しない。そのため、使用するカーネルごとに相対オフセットを事前に取得しておくことで、KASLR が有効な環境でも tcp_hashinfo の仮想アドレスを特定することができる。

4.4 アドレス変換

SVSM 内の追跡機構はゲスト OS のカーネルメモリを解析するために、ゲスト OS で用いられている仮想アドレスを物理アドレスへ変換し、対応するメモリページを SVSM のアドレス空間にマッピングして参照する。そのために、ゲスト OS のページテーブルを特定し、ページテーブルをたどることによって仮想アドレスを物理アドレスに変換する。変換後の物理アドレスに対応するページを SVSM の

表 1: 実験環境

		TDX 環境	SEV-SNP 環境
ホスト	CPU	Intel Xeon Gold 6526Y	AMD EPYC 9654
	メモリ	128 GiB	128 GiB
	NIC	Broadcom BCM57416	Broadcom BCM57416
	OS	Linux 6.8	Linux 6.11
ゲスト	vCPU	16	16
	メモリ	8 GiB	8 GiB
	NIC	virtio-net	virtio-net
	OS	Linux 6.12	Linux 6.11

```

root@guest:~# python3 tcp_conn_test.py
TCP connection established
[SVSM] guest tcp: bucket=27697 sock=0xffff888102db1280 state=1 127.0.0.1:12345 -> 127.0.0.1:45714
[SVSM] guest tcp: bucket=48207 sock=0xffff888102db3780 state=1 127.0.0.1:45714 -> 127.0.0.1:12345

```

図 3: 接続確立状態の TCP 接続情報の取得

仮想アドレス空間に一時的にマッピングし、その内容を読み出す。読み出すデータがページ境界をまたぐ場合には、ページごとに分割して同様の処理を行う。

ゲスト OS のページテーブルを特定する方法は TDX-SVSM と SEV-SVSM で異なる。TDX-SVSM では、L2 VM において VM Exit が発生した際に保存される実行コンテキストから CR3 レジスタの値を取得する。このレジスタには L2 VM で動作するゲスト OS のページテーブルの物理アドレスが格納されている。一方、SEV-SVSM では、ゲスト OS から SVSM を呼び出した際にゲスト OS のレジスタが VM Save Area (VMSA) と呼ばれるメモリ領域に保存されるため、その中の CR3 レジスタの値を取得する。

5. 実験

SockTrawl による TCP 接続の追跡能力と、機密 VM 内に隔離環境および追跡機構を導入することによるオーバーヘッドを調べる実験を行った。この実験は表 1 に示す Intel プロセッサと AMD プロセッサを搭載した 2 台のサーバを用いて行い、TDX 環境と SEV-SNP 環境それぞれ SVSM および SockTrawl の導入による性能を比較した。

5.1 TCP 接続情報の取得

機密 VM 内で動作させたクライアントとサーバの間で TCP 接続を確立し、SVSM において TCP 接続情報を取得した。その際に、SVSM 内の追跡機構が出力したログを図 3 に示す。TCP の状態、送信元・宛先 IP アドレスおよびポート番号が出力されていることが分かる。この結果より、TDX-SVSM と SEV-SVSM の両方で接続確立状態の TCP ソケットから接続情報を取得できることを確認した。

次に、機密 VM 内で動作させたクライアントから同じ VM 内のサーバと外部サーバに対して HTTP 通信を行い、SVSM において TCP 接続情報を取得した。その際に、SVSM 内の追跡機構が出力したログを図 4 に示す。この結

```

root@guest:~# python3 -m http.server 8080 &
[1] 1269
root@guest:~# Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...

root@guest:~# curl -s localhost:8080 > /dev/null
127.0.0.1 - - [12/Aug/2026 11:11:22] "GET / HTTP/1.1" 200 -
root@guest:~# [SVSM] guest tcp: bucket=12966 sock=0xffff88818f27db00 state=6 127.0.0.1:8080 -> 127.0.0.1:43868
[SVSM] guest tcp: bucket=20862 sock=0xffff88818f27c200 state=6 127.0.0.1:8080 -> 127.0.0.1:48676
curl -s local
root@guest:~# curl -s example.com > /dev/null
root@guest:~# [SVSM] guest tcp: bucket=5866 sock=0xffff8881861d9600 state=6 10.0.2.15:41446 -> 172.66.147.243:80

```

図 4: TIME_WAIT 状態の TCP 接続情報の取得

果より, TDX-SVSM と SEV-SVSM の両方で TIME_WAIT 状態の TCP ソケットから接続情報を取得できることを確認した。

これらの実験より, VM 外にパケットが送信されない場合でも TCP 接続情報を取得できることが分かる。従来の SEV-tracker では, ユーザ VM 外にパケットが送信される場合しか通信を捕捉することができなかった。

5.2 接続追跡率

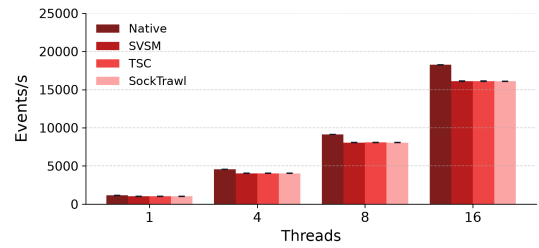
短時間に多数の TCP 接続を確立・切断した場合の接続追跡率を調べた。機密 VM 内のクライアントから同じ VM 内のサーバおよび外部サーバに対して, それぞれ 1000 回の HTTP 通信を行った。各 HTTP 通信は異なる curl プロセスを用いて実行し, SVSM 内の追跡機構が出力した TCP 接続情報を集計した。TDX-SVSM と SEV-SVSM のそれぞれについて, 追跡周期を 30 秒および 60 秒に設定して実験を行った。

その結果, いずれの場合も TIME_WAIT 状態の TCP ソケットから 1000 接続の情報を取得することができた。0~3 接続については, 接続確立状態の TCP ソケットからも情報を取得できた。追跡周期を 60 秒とした場合には, TIME_WAIT 状態の TCP ソケットの保持時間を超えてから追跡処理が実行される可能性があるが, 今回の実験では確認されなかった。

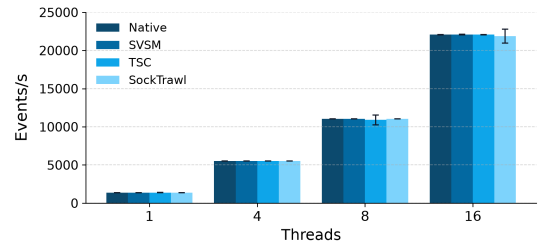
5.3 オーバヘッド

機密 VM 内に隔離環境を導入することによるオーバーヘッドと, SockTrawl の追跡機構によるオーバーヘッドを測定した。TDX と SEV-SNP それぞれについて, SVSM を用いない機密 VM (Native), SVSM を動作させた機密 VM (SVSM), SVSM において周期判定を行う機密 VM (TSC), SVSM において TCP 通信追跡を有効にした機密 VM (SockTrawl) の 4 つの環境を用意した。Native と SVSM を比較することで機密 VM 内に隔離環境を導入することによるオーバーヘッドを調べ, SVSM, TSC, SockTrawl を比較することで通信追跡によるオーバーヘッドを調べた。SockTrawl の追跡周期は 30 秒とし, ログ出力の影響を除くため, SVSM からシリアルコンソールへのログ出力を無効にした。

まず, sysbench 1.0.20 を用いて CPU 性能の測定を行った。この実験では, 素数計算の上限を 20,000 として 30 秒間実行し, 1, 4, 8, 16 スレッドを用いた場合の 1 秒あたり



(a) TDX



(b) SEV-SNP

図 5: CPU 処理性能

のイベント数を測定した。各条件について 3 回測定し, 平均値と 95%信頼区間を求めた。CPU 性能の測定結果を図 5 に示す。実験の結果, 隔離環境の導入により TDX 環境では性能が 12%低下することが分かった。一方, SEV-SNP 環境では性能低下はほとんど見られなかった。いずれの環境でも TSC を用いた周期判定と TCP 接続情報の取得が CPU 性能に与える影響は小さかった。

同様に, sysbench を用いてメモリ性能の測定を行った。この実験では, 各スレッドのローカルメモリ上にある 64MiB の領域をシーケンシャルに読み書きし, CPU 性能の測定と同じ 4 通りのスレッド数で 30 秒間のスループットを測定した。メモリ性能の測定結果を図 6 に示す。実験の結果, 隔離環境の導入により TDX 環境では性能が 6.5~9.6%低下した。一方, SEV-SNP 環境では性能が低下する条件もあったが, ほとんどの条件で性能が向上し, 最大で 5.6% 向上した。一方, 追跡機構のオーバーヘッドは TDX 環境ではわずかであったが, SEV-SNP 環境では性能が 5.3% の向上から 4.9% の低下まで, より大きく増減した。また, TDX 環境ではスレッド数に応じて性能が向上したのに対し, SEV-SNP 環境では 4 スレッドで飽和した。

次に, 追跡対象となる TCP 通信を行っている時の性能を調べるために HTTP サーバの性能を測定した。機密 VM 内で nginx を動作させて 1KiB のファイルを配信し, 外部ホストから wrk を用いて 30 秒間 HTTP リクエストを送信した。同時接続数は 1, 32, 128, wrk のスレッド数は 1, 4, 8, 16 とし, スレッド数が同時接続数以下となる組み合わせについて 1 秒あたりのリクエスト処理数を測定した。HTTP リクエスト処理性能の測定結果を図 7 に示す。隔離環境の導入により, TDX 環境では同時接続数が 1 および 32 の場合に 19~41% の大幅な性能低下が見られたが,

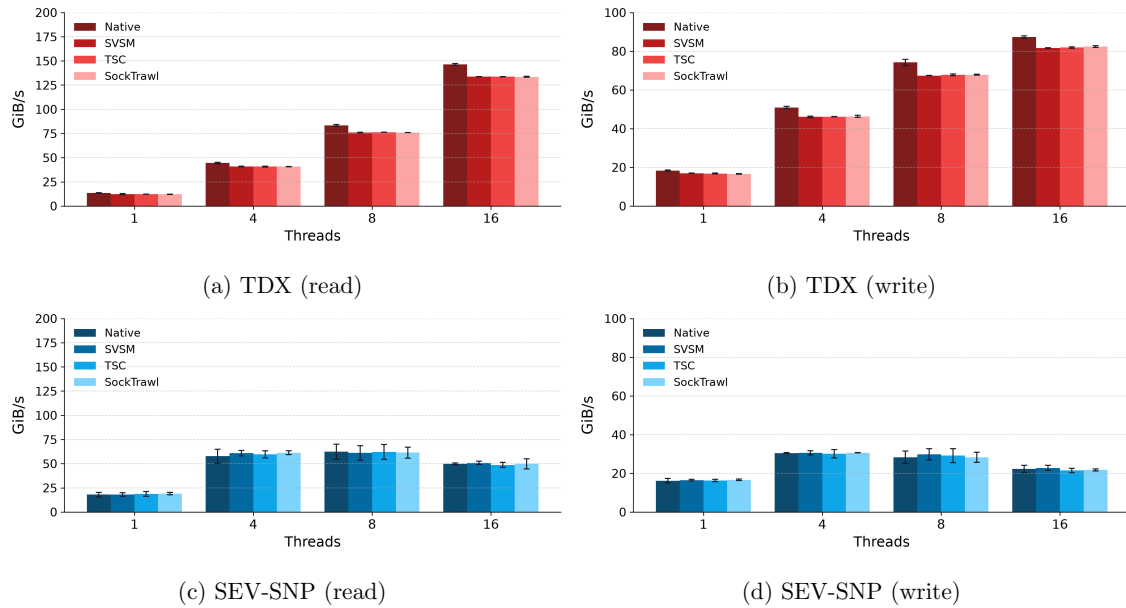


図 6: メモリ処理性能

同時接続数が 128 の場合にはほとんどの条件で性能が向上し、最大で 7.7% 向上した。一方、SEV-SNP 環境では、同時接続数が 1 および 32 の場合にはほとんど影響がなかったが、同時接続数が 128 の場合にはほとんどの条件で性能が低下し、最大で 10% の低下が見られた。追跡機構のオーバーヘッドは、TDX 環境では 4.2% の低下から 2.7% の向上まで、SEV-SNP 環境では 2.3% の低下から 17% の向上まで、測定条件によって増減した。

TDX 環境で隔離環境を導入した際に性能が低下した原因を調べるために、SVSM への遷移の頻度を測定した。機密 VM を 10 分間アイドル状態にし、その間に発生した VM Exit および Guest Exit の回数を計測した。1 秒あたりの平均遷移回数は、SEV-SVSM では 0.1 回であったのに対し、TDX-SVSM では 381 回であった。TDX-SVSM では SEV-SVSM よりも非常に高い頻度で SVSM への遷移が発生しており、この VM Exit の処理が TDX 環境における隔離環境のオーバーヘッドの要因の一つであると考えられる。

6. 関連研究

SEV-tracker [1] は、機密 VM とネストした仮想化を用いてクラウドサービスの通信を安全に追跡するシステムである。機密 VM 内でユーザ・ハイパーバイザとして BitVisor を動作させ、その上のユーザ VM でクラウドサービスを実行する。BitVisor がユーザ VM の通信パケットを捕捉することで、クラウドから保護された環境で通信を追跡する。しかし、ユーザ VM を動作させるためにネストした仮想化が必要であり、さらに通信ごとにパケットを捕捉して解析する必要がある。SockTrawl は機密 VM 内の隔離環境からカーネルメモリを直接解析することで、ネストした仮想化とパケットキャプチャを用いずに TCP 通信を追跡するこ

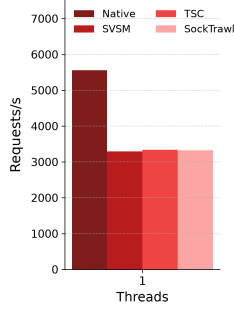
とができる。ただし、SEV-tracker ではユーザ VM を機密 VM として動作させることによってクラウドサービスを追跡機構から保護することができるのに対して、SockTrawl ではクラウドサービスは保護されない。

00SEVen [12] は、AMD SEV-SNP の VM Privilege Level (VMPL) を用いて機密 VM に VM イントロスペクションを適用することを可能にするシステムである。最も高い特権レベルの VMPL0 でエージェントを動作させ、ゲスト OS を低い特権レベルの VMPL で動作させることでエージェントをゲスト OS から隔離する。エージェントから VM のメモリやレジスタを取得することができるほか、VM の停止やメモリアクセスのトラップなども行うことができる。00SEVen では外部からの要求によって VM イントロスペクションを行うのに対し、SockTrawl では機密 VM 内の追跡機構が自律的に TCP 接続情報を取得する点が異なる。

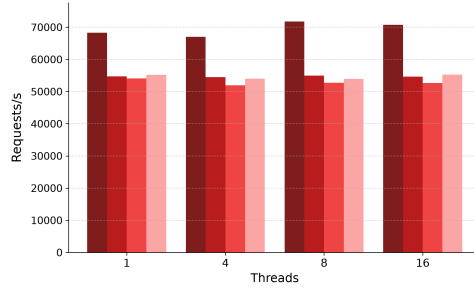
kdMonitor [13] は、Linux カーネルメモリ内の特権情報を監視し、権限昇格攻撃を検出するシステムである。監視対象の特権情報を専用のカーネルページに格納し、その内容を定期的に出力する周期監視と、ページへの書き込みをトリガとして出力する動的監視を行う。SockTrawl もカーネルメモリ上の情報を周期的に取得するが、ゲスト OS 内に監視用のカーネルモジュールを導入せず、ゲスト OS から隔離された SVSM において TCP ハッシュテーブルを解析する点が異なる。

7. まとめ

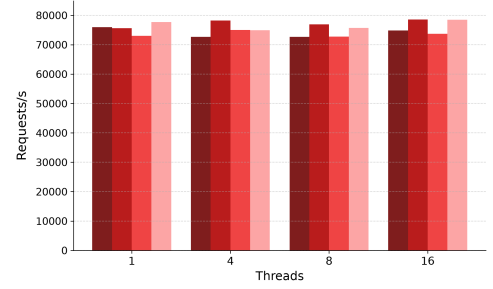
本稿では、機密 VM 内の隔離環境において Linux カーネルのメモリを解析することにより、TCP 通信を安全かつ軽量に追跡する SockTrawl を提案した。SockTrawl は TDX の TD Partitioning および SEV-SNP の VMPL を用



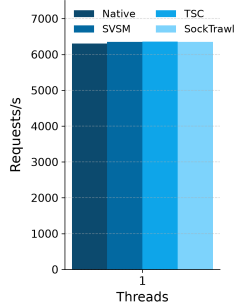
(a) TDX (同時接続数 1)



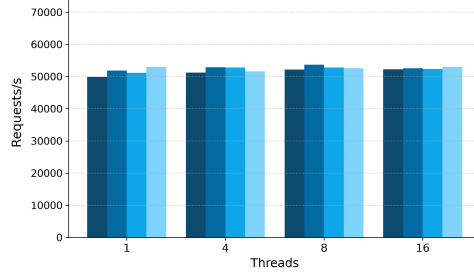
(b) TDX (同時接続数 32)



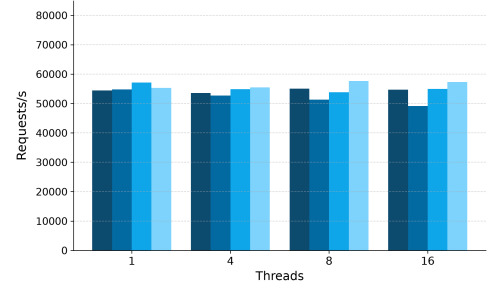
(c) TDX (同時接続数 128)



(d) SEV-SNP (同時接続数 1)



(e) SEV-SNP (同時接続数 32)



(f) SEV-SNP (同時接続数 128)

図 7: HTTP リクエスト処理性能

いて機密 VM 内の追跡機構をクラウドサービスから隔離する。追跡機構がゲスト OS の TCP ハッシュテーブルを定期的に走査することで TCP 接続情報を取得する。これにより、Nested SEV やパケットキャプチャを用いずにクラウドサービスの通信の追跡を可能にする。SockTrawl を TDX-SVSM および SEV-SVSM に実装して実験を行った結果、TCP 接続を 100% 追跡できることを確認した。また、SVSM のみを動作させた場合と比較して、通信追跡によるオーバーヘッドは小さいことを確認した。

今後の課題は、より安定かつ軽量の追跡を実現することと、取得した通信ログを安全にユーザに提供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] N. Ando, K. Takiguchi, and K. Kourai: Secure Privacy Control inside Clouds with AMD SEV and Nested Virtualization, 49th IEEE International Conference on Computers, Software, and Applications (COMPSAC), pp. 83–88 (2025).
- [2] K. Takiguchi and K. Kourai: Nested SEV: Secure and Generic SEV Support for Nested Virtualization, 20th USENIX Symposium on Operating Systems Design and Implementation, pp. 1051–1066 (2026).
- [3] Intel: Intel TDX Module Partitioning Architecture Specification (online), <https://cdrdv2.intel.com/v1/dl/getContent/773039> (2024).
- [4] COCONUT Secure VM Service Module (online), <https://github.com/coconut-svsm/svsm> (2022).
- [5] P. Fang et al.: Intel TD Partitioning and vTPM on COCONUT-SVSM (online), https://lpc.events/event/18/contributions/1918/attachments/1632/3406/02-lpc2024_mc_tdp_vtpm.pdf (2025).
- [6] Intel: td-partitioning-svsm (online), <https://github.com/intel-staging/td-partitioning-svsm> (2025).
- [7] Intel: Intel Trust Domain Extensions (Intel TDX) (online), <https://cdrdv2.intel.com/v1/dl/getContent/690419> (2025).
- [8] AMD: AMD Secure Encrypted Virtualization (SEV) (online), <https://www.amd.com/ja/developer/sev.html>.
- [9] Arm: Arm Confidential Compute Architecture (online), <https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture>.
- [10] BitVisor (online), <https://www.bitvisor.org/>.
- [11] S. Kuenzer et al.: Unikraft: Fast, Specialized Unikernels the Easy Way, Proc. European Conf. Computer Systems, pp. 376–394 (2021).
- [12] F. Schwarz and C. Rossow: 00SEVen – Re-enabling Virtual Machine Forensics: Introspecting Confidential VMs Using Privileged in-VM Agents, 33rd USENIX Security Symposium (USENIX Security 24) (2024).
- [13] H. Kuzuno and T. Yamauchi: Identification of Privilege Escalation Attack Using Kernel Monitoring Mechanism, Journal of Information Processing, Vol. 33, pp. 1143–1153 (2025).