eBPFプログラムによる安全かつ透過的なVM内システム監視機構

堀 恭介[1]　光来 健一[1]

概要：
Amazon EC2 などの IaaS 型クラウドはユーザに仮想マシン（VM）を提供しており，クラウドは VM 内で動作するエージェントを用いて情報を収集することが多い．エージェント方式の問題点は VM のユーザがエージェントの保守作業を怠ると脆弱性となる可能性があることである．それに対して，イントロスペクション方式はクラウドが VM のメモリなどに直接アクセスして情報を取得できるが，AMD SEV などによりメモリが暗号化された VM には適用できない，本稿では，クラウドが VM に eBPF プログラムを動的に送り込み，VM 内の情報を安全かつ透過的に取得するシステム TeleBPF を提案する．TeleBPF では，eBPF アプリケーションに提供する TeleBPF 共有ライブラリが eBPF 関連システムコールを横取りし，VM 専用の通信機構を用いて VM 内の TeleBPF プロキシに転送して代理実行する．また，eBPF アプリケーションが VM 内のリングバッファのメモリに直接アクセスすることにより，eBPF プログラムから情報を高速に取得することができる．TeleBPF を用いて既存の eBPF アプリケーションが実行できることを確認し，仮想化のオーバヘッド削減により高速化されることが分かった．

1. はじめに

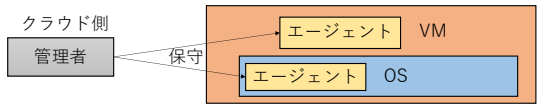
Amazon EC2 などの IaaS 型クラウドは仮想マシン（VM）を提供しており，ユーザは VM 内のシステムを自由に管理することができる．一方，クラウドは VM 内のシステム情報を取得することにより，VM のセキュリティを向上させるための監視サービスを提供することができる．クラウドによる VM の監視手法として，エージェント方式が一般的に用いられている．この方式は VM 内にエージェントと呼ばれるソフトウェアを組み込み，クラウドはエージェントと通信して情報を取得する．エージェントは OS のプロセスとして実行する場合とカーネルモジュールとして実行する場合があり，前者の例として Amazon CloudWatch エージェント [1]，後者の例として IBM Cloud のモニタリング・エージェント [2] が挙げられる．

エージェント方式の問題点は，ユーザがエージェントのインストールやバージョンアップなどの保守作業を行う必要があることである．この作業を怠るとエージェントが VM 内のシステムの脆弱性となる可能性がある．また，エージェントをプロセスとして実行すると取得できない情報があり，カーネルモジュールとして実行するとシステムが不安定になる可能性がある．この問題を解決するために，VM のメモリなどに直接アクセスして情報を取得するイントロスペクション方式 [3] が用いられている．しかし，低レベルなデータ構造の解析が必要となり，監視システムの開発が難しい．また，AMD SEV [4] などによって VM のメモリが暗号化されていると利用することができない．

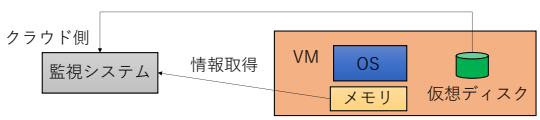
本稿では，クラウドから VM に eBPF プログラムを送り込んで実行し，VM 内のシステムを安全かつ透過的に監視するシステム TeleBPF を提案する．eBPF は Berkeley パケットフィルタを拡張した Linux の機構であり，システム内でのイベント発生時など，指定した箇所でシステムの状態を監視することができる．TeleBPF はクラウド側で既存の eBPF アプリケーションを透過的に実行し，VM 内の OS に eBPF プログラムをロードしたり，送り込んだ eBPF プログラムから情報を取得したりすることを可能にする．eBPF プログラムを VM 内に動的にロードすることにより，ユーザによる保守作業が不要になり，AMD SEV によるメモリ暗号化の影響も受けない．また，eBPF プログラムは実行時に検査器を用いてチェックされるため OS 内で安全に実行することができ，開発も比較的容易である．

TeleBPF は eBPF アプリケーションに TeleBPF 共有ライブラリを提供し，eBPF 関連システムコールを横取りして VM 専用の通信機構を用いて VM に転送する．システムコール呼び出しの横取りは Zpoline [5] を用いてバイナリ書

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



(a) エージェント方式



(b) イントロスペクション方式

図 1: 既存の情報取得手法

き換えを行うことにより実現する．転送したシステムコールは VM 内で動作する TeleBPF プロキシが VM 内の OS に対し代理で実行し，実行結果を TeleBPF 共有ライブラリ経由で eBPF アプリケーションに返す．eBPF アプリケーションが VM のメモリに直接アクセスすることにより，VM 内の eBPF プログラムから情報を取得するために用いられるリングバッファにも高速にアクセスすることができる．TeleBPF を用いて Microsoft Sysmon for Linux [7] が動作することを確認し，仮想化のオーバヘッドが削減されることにより VM 内で実行するよりも高速化されることが分かった．

以下，2 章では従来のエージェント方式とイントロスペクション方式について説明し，その問題点を述べる．3 章では eBPF を用いた新たな情報取得手法である TeleBPF を提案する．4 章では TeleBPF 共有ライブラリと TeleBPF プロキシを用いた TeleBPF の実装について説明する．5 章では TeleBPF の動作とオーバヘッドを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 章で本稿をまとめる．

2. VM 内の情報取得

IaaS 型クラウドにおいて VM はユーザが管理しているが，クラウド側も VM 内の情報を取得して活用している．例えば，VM 内のシステムの情報を用いることで，クラウド側から VM への侵入を安全に検知することができる．クラウドによる情報取得方法としては，**図 1a** のように VM 内にエージェントと呼ばれるソフトウェアを組み込み，クラウド側がエージェントと通信して情報を取得するエージェント方式がよく用いられている．エージェントは OS のプロセスとして実行する場合とカーネルモジュールとして実行する場合がある．前者の例として Amazon CloudWatch エージェント [1] が挙げられ，VM 内のシステムのログやメトリクスの収集，ログやトレースの分析などを行うために用いられている．一方，後者の例としては IBM Cloud のモニタリング・エージェント [2] が挙げられ，実行されたシステムコールの情報も収集することができる．

エージェント方式の問題点はエージェントの管理を VM のユーザが行う必要があることである．導入時にインストール作業を行い，その後も定期的にアップデートを行う必要がある．保守作業を怠った場合，VM 内のシステムは堅牢であったとしても，エージェントが脆弱性となり外部からの攻撃を受ける可能性がある．また，エージェントをプロセスとして実行すると取得できない情報が存在し，侵入者に無効化されやすい．一方，カーネルモジュールとして実行するとカーネルに組み込まれるため攻撃は受けにくくなるが，システムが不安定になる可能性がある．さらに，OS のアップデートに合わせてエージェントも常に最新にしておかなければ動作しなくなる可能性がある．

一方，**図 1b** のように VM の外部から VM のメモリや仮想ディスクなどに直接アクセスして解析を行うことで情報を取得するイントロスペクション方式も提案されている [3]．例として，VM のメモリ上にある OS のデータ構造を解析することにより，セキュリティに関連する情報を取得することができる．また，VM の仮想ディスクで用いられているファイルシステムを解析することにより，ファイルの検査を行ったり設定ファイルを調べたりすることができる．この方式はエージェント方式と異なり，VM 内へのソフトウェアのインストールやその保守作業が必要ないという利点を持つ．

しかし，イントロスペクション方式にも様々な問題点がある．VM のメモリや仮想ディスクなどの低レベルな解析を行う必要があるため，監視システムの開発が難しい．特に，メモリ上の OS のデータ構造は OS のバージョンに大きく依存するため，OS の開発に合わせて監視システムの開発を行う必要がある．さらに，AMD SEV や Intel TDX などを用いて VM のメモリが暗号化されていると，クラウド側から VM のメモリを解析することはできない．また，VM 内の OS が仮想ディスクを暗号化していると，クラウド側から仮想ディスクを解析することもできない．

3. TeleBPF

本稿では，クラウド側から VM に eBPF プログラムを送り込んで実行し，VM 内のシステムを安全かつ透過的に監視するシステム TeleBPF を提案する．eBPF フレームワークでは，eBPF アプリケーションが eBPF プログラムを OS にロードすることにより，OS 内の様々な情報を取得することができる．TeleBPF では**図 2** のように，クラウド側で既存の eBPF アプリケーションを実行し，VM 内の TeleBPF プロキシを経由して eBPF プログラムを VM 内の OS にロードする．eBPF プログラムが取得した情報は TeleBPF プロキシ経由でクラウド側の eBPF アプリケーションに返される．TeleBPF プロキシは監視対象システムをコンテナに隔離することにより保護する．このように，TeleBPF は eBPF プログラムを高機能かつ安全なエージェントとして用いる．

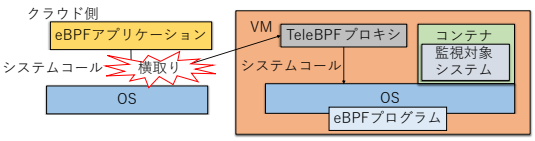


図 2: TeleBPF のシステム構成

従来のエージェント方式と比べて，TeleBPF は eBPF プログラムを VM 内に動的に送り込むことができる．VM のユーザはエージェントをインストールする必要も保守作業を行う必要もない．eBPF プログラムは OS 内で実行されるため攻撃の影響を受けにくく，プロセスとして実行されるエージェントよりも多くの情報を取得できる．eBPF プログラムはロード時に検査器を用いて検査されるため，OS カーネル内で実行してもシステムに悪影響を及ぼすことはない．一方，イントロスペクション方式と異なり，TeleBPF は eBPF プログラムを VM 内で実行するため，VM のメモリ暗号化の影響を受けずに情報を取得することができる．また，低レベルな VM のメモリ解析も不要である．

クラウド側で eBPF アプリケーションを透過的に実行できるようにするために，TeleBPF は eBPF 関連システムコールを VM 内の TeleBPF プロキシに転送する．eBPF 関連システムコールには 3 種類あり，eBPF 制御システムコールは eBPF プログラムに関連するコマンドを実行するために用いられる．例えば，eBPF プログラムのロードを行うコマンドや，eBPF プログラムと eBPF アプリケーションの間でデータをやりとりするための BPF マップに関するコマンドなどがある．イベント制御システムコールはシステム内のイベントを eBPF プログラムに関連づけるために用いられる．例えば，カーネル内の特定の関数が実行された時に eBPF プログラムを呼び出すことができる．eBPF のために用いられる特殊ファイルを扱うシステムコールは汎用的なファイル関連システムコールであり，VM 内のトレース情報を取得したりするために用いられる．

TeleBPF は eBPF 関連システムコールを以下のように VM に転送する．まず，eBPF アプリケーションにおいてシステムコールをフックする．eBPF 制御システムコールは必ず VM 内で実行する必要があるため，常に VM に転送する．イベント制御システムコールには eBPF に関連しない処理を行うものも含まれるため，システムコールによっては引数で転送するかどうかを判断する．特殊ファイルを扱うシステムコールはすべて汎用的なシステムコールであるため，引数で転送するかどうかを判断する．システムコールの引数はシリアライズしてバイト列に変換して転送する．転送したシステムコールは TeleBPF プロキシが代わりに実行し，返り値を含む実行結果を eBPF アプリケーションに返送する．

eBPF アプリケーションは eBPF プログラムからの情報取得にリングバッファを利用する場合がある．リングバッファは eBPF アプリケーションとカーネル内の eBPF プログラムの間の共有メモリ上に作成される．リングバッファへのアクセスはシステムコールを用いずに行われるため，eBPF アプリケーションによるアクセスを VM に転送するのは難しい．メモリアクセスをトラップして転送することは可能ではあるが，大幅な性能低下は避けられない．そこで，TeleBPF は VM 内のリングバッファのメモリをクラウド側と共有することで，eBPF アプリケーションが直接，リングバッファにアクセスできるようにする．そのために，カーネル内のリングバッファのメモリを eBPF アプリケーションにマップするためのシステムコールを VM に転送し，返り値のアドレスに対応する VM のメモリを eBPF アプリケーションからアクセスできるようにする．

4. 実装

KVM を用いて作成された VM 上で動作する Linux 5.15 に対して TeleBPF を実装した．

4.1 システムコールのフック

eBPF アプリケーションによって呼び出されるシステムコールをフックするために，TeleBPF は Zpoline [5] と呼ばれる機構を用いる．Zpoline は eBPF アプリケーションが利用しているライブラリも含めてシステムコールの呼び出し箇所をすべて実行時に書き換え，TeleBPF 共有ライブラリで定義した関数を呼び出す．フックする必要があるシステムコールについては，その関数内で VM への転送処理を行う．Zpoline を用いることで，従来の ptrace を用いる手法よりも高速なフックを実現することができる．一方，LD_PRELOAD 環境変数を用いて実行時にシステムコール関数を置き換える手法はより高速であるが，システムコール関数がインライン関数の場合にはその関数を実行するすべての関数を置き換える必要がある．これはライブラリの実装に依存し，網羅的にすべての関数を置き換えるのは難しい．

4.2 システムコールの転送

システムコールをフックした TeleBPF 共有ライブラリはまず，システムコール番号を VM 内の TeleBPF プロキシに転送する．次に，プロトコルバッファ [10] を用いてシステムコールの引数をシリアライズして転送する．TeleBPF プロキシでは，システムコール番号に応じて，送られてきた引数をデシリアライズする．そして，eBPF アプリケーションの代わりにシステムコールを実行し，返り値を返す．システムコールの実行に失敗した場合にはエラー番号も返す．また，必要に応じてシステムコールがカーネルから受け取ったデータも返す．システムコールを VM に転送する際には，VM 専用の通信機構である VM ソケット (Vsock) [11] を用いる．Vsock はホストと VM の間の通信

に用いられ，TCP/IP を用いるよりも高速に通信を行うことができる．

TeleBPF プロキシはシステムコールが返すファイル記述子の値を変換して，ファイル記述子を引数にとるシステムコールを転送するかどうか判別できるようにする．TeleBPF プロキシが代理で実行したシステムコールは返り値や引数でファイル記述子を返す場合があるが，この値は TeleBPF プロキシの中でのみ一意に識別可能である．このファイル記述子をそのまま eBPF アプリケーションに返すと，eBPF アプリケーションの中で使われているものと区別できなくなる．そのため，dup2 システムコールを用いてファイル記述子を複製し，元のファイル記述子に一定の値を足したファイル記述子を作成する．このファイル記述子を eBPF アプリケーションに返すことで，eBPF アプリケーション内でも一意に識別可能にする．

4.3 bpf システムコールの転送

eBPF アプリケーションが eBPF 制御システムコールである bpf システムコールを呼び出すと，TeleBPF 共有ライブラリがそれをフックして VM に転送する．bpf システムコールには引数で様々な eBPF コマンドが指定され，それぞれのコマンドが引数の bpf_attr 共用体の異なるメンバを使用する．そのため，TeleBPF プロキシには同一のシステムコール番号ではなく，コマンドごとに異なる番号を転送して区別する．また，bpf_attr 共用体については必要なメンバだけを転送する．TeleBPF プロキシは受信したデータから bpf_attr 共用体を作成し，bpf システムコールを実行する．そして，システムコールの返り値を TeleBPF 共有ライブラリに返し，それが eBPF アプリケーションに返される．

例えば，eBPF プログラムをロードする eBPF コマンドの場合，bpf_attr 共用体に格納されている eBPF プログラムの種類，名前，バイトコードなどの情報を転送する．返り値として，ロードした eBPF プログラムにアクセスするためのファイル記述子を返す．eBPF プログラムをトレースポイントにアタッチする eBPF コマンドの場合，bpf_attr 共用体に格納されている eBPF プログラムのファイル記述子とトレースポイント名を転送する．一方，BPF マップを操作する eBPF コマンドの場合，BPF マップを作成するコマンドでのみ bpf_attr 共用体においてキーとバリューのサイズが指定される．BPF マップのデータを参照・更新・削除する際にキーやバリューを転送できるようにするために，作成時に TeleBPF 共有ライブラリと TeleBPF プロキシの双方でキーとバリューのサイズを保存しておく．BPF マップを参照する際にはシステムコールが bpf_attr 共用体にバリューを格納するため，TeleBPF プロキシは返り値とともにバリューを返送する．

4.4 イベント制御システムコールの転送

イベント制御システムコールにはイベントに応じて多種多様なものが存在するため，その中のいくつかについて説明する．perf_event_open システムコールは監視のためのイベントを設定するために用いられる．引数の perf_event_attr 構造体にはイベントの種類，監視する関数のアドレス，トレースポイントの ID などのイベントを指定するために必要な情報が格納されている．eBPF アプリケーションがこのシステムコールを呼び出すと，TeleBPF 共有ライブラリがそれをフックして VM に転送し，イベントにアクセスするためのファイル記述子を返り値として受け取る．

eBPF アプリケーションはこのファイル記述子を引数に指定して ioctl システムコールを呼び出すことにより，イベントへの eBPF プログラムの関連づけや有効化，無効化を行う．TeleBPF 共有ライブラリはこのシステムコール呼び出しをフックするが，ioctl システムコールは様々な用途に利用される．perf_event_open システムコールの返り値のファイル記述子に TeleBPF プロキシが一定値を足しているため，引数のファイル記述子が一定値を超えている場合にのみ VM に転送する．ioctl 関数は第 3 引数として任意の型のデータを取るため，第 2 引数で指定されたコマンド番号に応じて適切なサイズのデータを転送する．

epoll 関連システムコールはファイル記述子を監視して入出力を待つためのものであり，リングバッファにデータが格納されたことの通知を受け取るために用いられる．epoll_create1 システムコールは epoll インスタンスを作成してファイル記述子を返す．epoll_ctl システムコールはリングバッファに対応するファイル記述子を epoll インスタンスに設定する．epoll_wait システムコールは epoll イベントが発生するのを待ち，イベント一覧を返す．これらのシステムコールはリングバッファ以外にも利用されるが，現在の実装では常に TeleBPF プロキシに転送している．eBPF アプリケーション側のイベントを待つ必要がある場合については今後の課題である．

4.5 ファイル関連システムコールの転送

特殊ファイルへのアクセスに用いるファイル関連システムコールの中で，openat システムコールや newfstatat システムコールなど，引数にパス名をとるものはパス名で転送の判別を行う．例えば，イベント追跡に関する特殊ファイルや CPU 数に関する情報を提供する特殊ファイルの場合には VM への転送を行う．これらのシステムコールはディレクトリを指定するファイル記述子も引数にとるため，このファイル記述子の値が一定値を超えている，つまり，TeleBPF プロキシがオープンしたものである場合には転送を行う．read システムコールなどはファイル記述子が一定値以上であれば転送し，TeleBPF プロキシは返り値に加えて特殊ファイルのデータも返送する．

表 1: 実験環境

	ホスト	VM
CPU	Intel Core i7-10700	1
メモリ	64GB	1GB
OS	Linux 5.15	Linux 5.15
仮想化ソフトウェア	8.0.0	-
プロトコルバッファ	3.12.4	3.12.4

4.6 mmap システムコールの転送

eBPF アプリケーションがカーネル内のリングバッファのメモリをマップするために mmap システムコールを呼び出すと，TeleBPF 共有ライブラリはそれをフックする．mmap システムコールは引数にファイル記述子をとるため，それが一定値以上，つまり，リングバッファのための BPF マップのものである場合だけ TeleBPF プロキシに転送する．VM 内の TeleBPF プロキシはカーネル内のリングバッファのメモリをマップしてそのアドレスを返送する．返されたアドレスは TeleBPF プロキシの仮想アドレスであるため，そのままでは TeleBPF 共有ライブラリはメモリにアクセスすることができない．

そこで，TeleBPF 共有ライブラリはまず，受け取った仮想アドレスを TeleBPF プロキシのページテーブルを用いて物理アドレスに変換する．このページテーブルのアドレスは TeleBPF プロキシに接続した際に取得しておく．TeleBPF プロキシはカーネルモジュールを用いてカーネルから取得する．TeleBPF 共有ライブラリが VM のメモリにアクセスできるようにするために，VM のメモリがファイルとして作成されるようにして VM を起動しておく．このファイルをメモリにマップしてページテーブルにアクセスする．リングバッファは複数のページで構成されるため，ページごとに先頭アドレスを物理アドレスに変換する．次に，それぞれの物理アドレスに対応するメモリファイルの領域をマップしていき，eBPF アプリケーションの仮想アドレス空間上で連続した領域になるようにする．最後に，その領域の先頭アドレスを eBPF アプリケーションに返す．

5. 実験

TeleBPF を用いて eBPF アプリケーションを実行し，VM に送り込んだ eBPF プログラムから情報を取得できることを確認した．また，TeleBPF のオーバヘッドを測定する実験を行った．比較として，TeleBPF を用いずに VM 内で eBPF アプリケーションを実行した場合についても測定を行った．この実験には表 1 のホストと VM を用いた．

5.1 動作確認

TeleBPF を用いて Microsoft Sysmon for Linux[7] を実行し，その動作を確認した．Sysmon はプロセスの作成・終了やネットワーク接続，ファイルの書き込みといったシステムの行動を監視し，ログとして記録するツールである．Sysmon は libbpf[12] を用いて C 言語で記述されている．この実験では未対応のネットワーク監視機能は無効化した．実験の結果，VM 内でコマンドを実行すると**図 3** のようにホスト上でプロセスに関するログが出力されることを確認した．

```
Event SYSMONEVENT_CREATE_PROCESS
        RuleName: -
        UtcTime: 2024-02-11 06:37:45.
        ProcessGuid: {c937e8a0-6b39-6
        ProcessId: 18000
        Image: /usr/bin/ls
        FileVersion: -
```

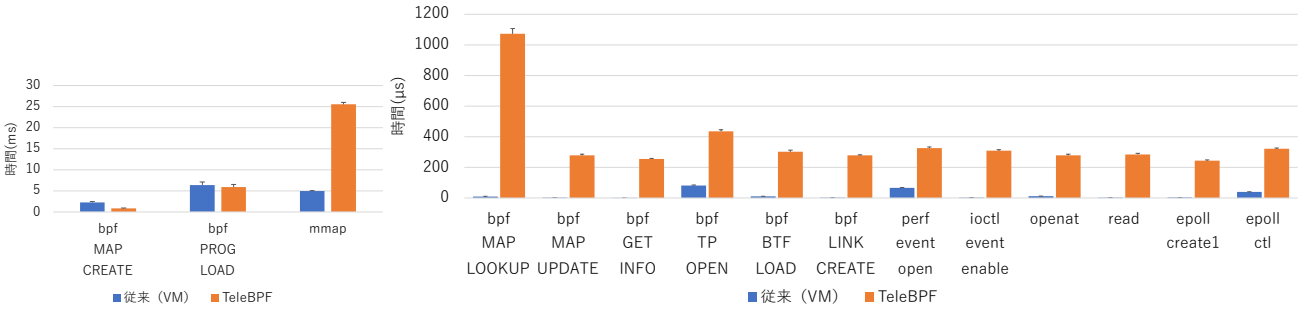
図 3: TeleBPF を用いて実行した Sysmon のログ出力

5.2 システムコール実行性能

TeleBPF を用いて Sysmon を実行した場合の eBPF 関連システムコールの実行時間を測定した．ただし，bpf システムコールについてはコマンドごとに測定した．また，ほとんどの epoll_wait システムコールの実行時間はタイムアウトするまでの 0.1 秒となるため，測定からは除外した．測定結果は**図 4** のようになった．四分位範囲の 1.5 倍を超える外れ値は除外している．TeleBPF でシステムコールを転送した場合には，VM 内で実行した場合と比べて mmap システムコールの実行時間が 15.8ms 長くなり，5.1 倍の時間がかかった．bpf システムコールの MAP_LOOKUP_ELEM コマンドでも実行時間が 1.1ms 長くなった．それ以外のほとんどのシステムコールでは実行時間が 241〜353μs 長くなった．一方，bpf システムコールの BPF_MAP_CREATE コマンドは逆に実行時間が 1.4ms 短くなった．

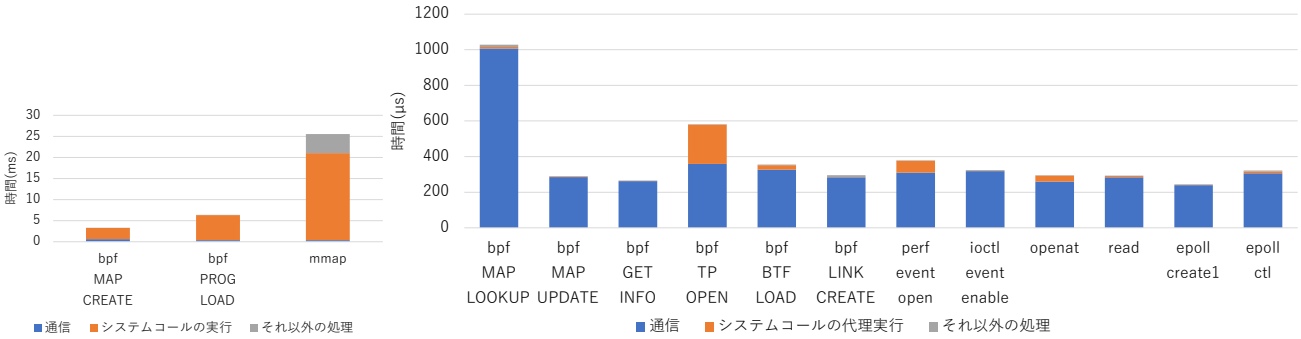
Sysmon の実行中に bpf システムコールの BPF_PROG_LOAD コマンドは 19 回実行され，様々なサイズの eBPF プログラムをロードした．また，BPF_BTF_LOAD コマンドは 9 回実行され，様々なサイズの型情報をロードした．そのため，システムコールの引数で渡されるこれらのデータのサイズが実行時間にどのような影響を及ぼすかについて調べた．その結果，データサイズが大きくなると実行時間も増える傾向にあったが，ばらつきが大きかった．

TeleBPF を用いた場合のシステムコール実行時間の内訳を調べるために，通信時間，プロキシでのシステムコールの代理実行時間，それ以外の処理にかかる時間に分けて測定を行った．測定結果を**図 5** に示す．全システムコールで平均すると，通信時間が 75.1%，システムコールの代理実行時間が 22.7%，それ以外の処理時間が 2.1%であることが分かった．これにより，通信時間がシステムコール実行時間の大きな割合を占めることが分かった．一方，bpf システムコールの BPF_MAP_CREATE コマンドと



(a) 実行時間の長いシステムコール　(b) 実行時間の短いシステムコール

図 4: システムコール実行時間



(a) 実行時間の長いシステムコール　(b) 実行時間の短いシステムコール

図 5: システムコール実行時間の内訳

BPF_PROG_LOAD コマンド，mmap システムコールではシステムコールの代理実行時間が占める割合の方が大きかった．特に，mmap システムコールは VM 内で Sysmon を実行した場合よりも，プロキシによる代理実行の方が 4.2 倍も時間がかかった．この原因は調査中である．また，mmap システムコールではそれ以外の処理時間の占める割合が 10%以上となった．これはリングバッファのサイズが 64MB あり，仮想アドレスを物理アドレスに変換したり，物理アドレスに対応するメモリファイルの領域をマップしたりする処理に時間がかかったためである．

Vsock を用いることによる通信の高速化について調べるために，TCP/IP を用いて TeleBPF プロキシと通信を行うようにした場合についてシステムコール実行時間を測定した．実験結果は**図 6** のようになり，Vsock を用いると平均で 10%高速になることが分かった．bpf システムコールの BPF_MAP_LOOKUP_ELEM コマンド，BPF_OBJ_GET_INFO_BY_FD コマンド，BPF_BTF_LOAD コマンドなどでは大幅に高速化した．

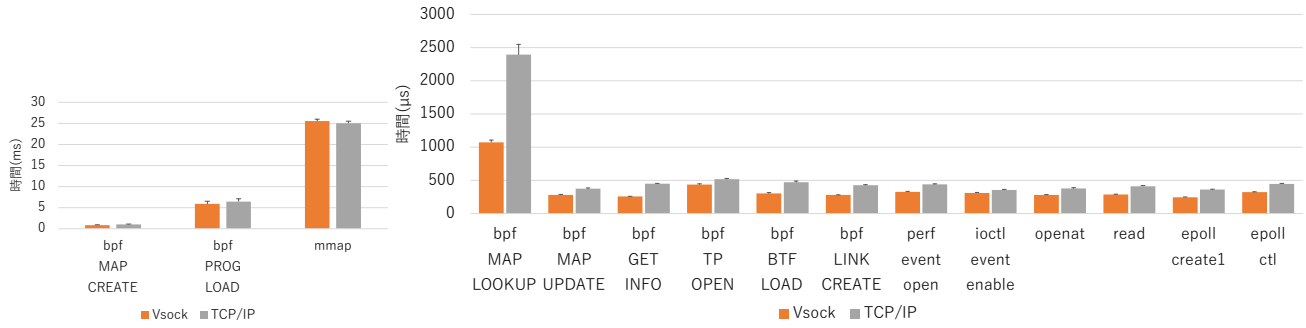
5.3 情報取得性能

eBPF アプリケーションは BPF マップまたはリングバッファを用いて eBPF プログラムから情報を取得する．BPF マップから取得する場合には bpf システムコールの BPF_MAP_LOOKUP_ELEM を実行し，リングバッファから取得する場合にはシステムコールは実行しない．TeleBPF を用いて BPF マップまたはリングバッファから情報を取得するのにかかる時間を測定した．比較として，通信に TCP/IP を用いた場合についても測定した．BPF マップからの情報取得時間は**図 7a** のようになった．実験結果より，TeleBPF で bpf システムコールを転送した場合には，VM 内で実行した場合と比べて情報取得時間が 199μs 長くなることが分かった．このオーバヘッドにより情報取得時間は 19 倍になった．

一方，リングバッファからの情報取得時間は**図 7b** のようになった．この結果より，TeleBPF を用いると VM 内よりも 1.8 倍高速に情報が取得できることが分かった．どちらの場合もリングバッファのメモリから直接，情報を取得する点では同じであるが，VM 内で情報を取得する場合には仮想化のオーバヘッドにより性能が低下したと考えられる．リングバッファでは情報取得の際に通信は行われないため，Vsock による高速化の効果はないはずであるが，TCP/IP を用いる場合より少しだけ高速になった．これは誤差だと考えられる．以上のことより，TeleBPF を用いる場合にはリングバッファを用いると通信のオーバヘッドなしで高速に情報を取得することができる．

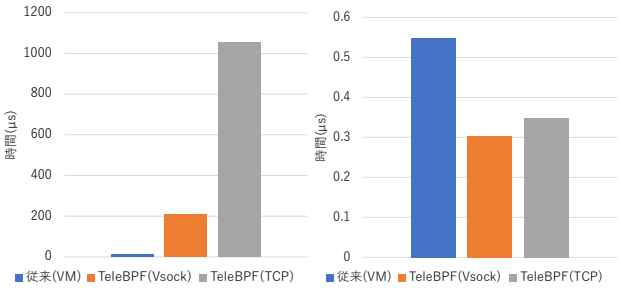
5.4 Sysmon の性能

Sysmon が eBPF 関連の初期化を行うのにかかる処理時



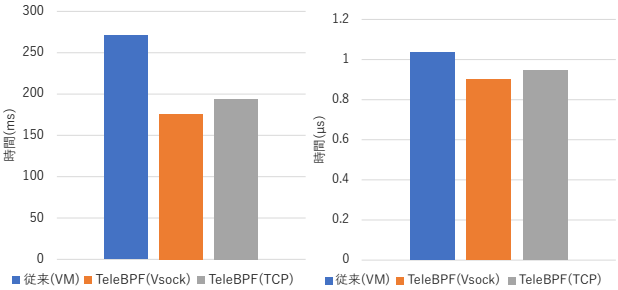
(a) 実行時間の長いシステムコール　(b) 実行時間の短いシステムコール

図 6: Vsock によるシステムコール実行の高速化



(a) BPF マップ　(b) リングバッファ

図 7: 情報取得時間



(a) 初期化処理　(b) 情報取得

図 8: Sysmon の実行時間

間を測定した．比較として，TeleBPF を用いずに VM 内で Sysmon を実行した場合と Vsock の代わりに TCP/IP を用いた場合についても測定を行った．初期化処理にかかる時間は**図 8a** のようになった．この結果より，TeleBPF を用いると VM 内で実行するより 54%高速になることが分かった．これは TeleBPF でシステムコールを転送するオーバヘッドよりも，Sysmon を VM 内で実行することによる仮想化オーバヘッドの方が大きいためと考えられる．一方，TCP/IP を用いる場合と比べて，Vsock を用いることにより 11%高速になることが分かった．システムコールの転送によるデータ転送レートは，Vsock を用いる場合は 978KB/s，TCP/IP を用いる場合は 885KB/s であった．

また，Sysmon が eBPF プログラムから情報を取得するのにかかる時間を測定した．VM 内で ls コマンドを実行し，その際に perf リングバッファから情報を取得するのにかかる時間を測定した．情報取得時間は**図 8b** のようになった．この結果より，TeleBPF を用いると VM 内で実行するより 13%高速になることが分かった．

6. 関連研究

BPFd [13] は BPF Compiler Collection (BCC) を用いて開発された eBPF アプリケーションに対して，リモートホストからの情報取得を可能にする．主に BCC をインストールできない Android 端末で BCC の機能を用いるために開発さた．eBPF プログラムのロードや BPF マップの作成などのコマンドをリモートホストの BPFd デーモンに転送して実行する．BPFd は TeleBPF と似ているが，BPFd では BCC ライブラリ内の関数呼び出しを転送しているため，BCC の実装の変更に影響されやすい．TeleBPF はシステムコールの呼び出し自体を転送するため，BCC の開発の影響は受けない．システムコールの仕様変更の頻度は低く，後方互換性が保たれることが多いため，TeleBPF の方が汎用性は高い．また，TeleBPF は BCC 以外の eBPF を用いるフレームワークでも利用可能である．

IBM Cloud のモニタリング・エージェントとしても使われている Sysdig [14] は従来，VM 内にカーネルモジュールをインストールすることでシステムコールの実行を監視していた．最近は eBPF プログラムをロードすることで同等の機能を実現している．安全に監視できる一方で性能が少し低下することから，カーネルモジュールの提供も続けられている．eBPF プログラムを使う場合でも，クラウドでは Sysdig を VM にインストールして利用するため，VM のユーザが eBPF プログラムを含めて Sysdig 全体の保守を行う必要がある．TeleBPF ではクラウド側が監視ツール全体の保守を行うことができる．

SEVmonitor [15] は SEV を用いてメモリが暗号化された VM 内で安全にエージェントを動作させることにより，侵入検知システム (IDS) のオフロードを可能にする．SEV で VM が保護されている場合，イントロスペクション方式

を用いてメモリ上のデータを監視することはできないが，エージェント経由でメモリデータを取得することによってVM の監視を行うことができる．VM 内のエージェントを保護するために，OS 内で動作させる手法とハイパーバイザ内で動作させる手法が提案されている．エージェントをOS 内で動作させる手法は TeleBPF に似ているが，eBPF プログラムを用いる TeleBPF の方が安全性は高い．

イントロスペクション方式を用いた監視ツールの開発を容易にする手法も提案されている．VMST [16] は監視対象VM と同一の OS がインストールされたセキュア VM 内で既存の監視ツールを動作させることができる．VMST はイントロスペクションが必要なデータを自動的に識別し，そのデータアクセスを監視対象 VM 内のカーネルメモリに転送する．LLView [17] は OS のソースコードを用いて作成された監視ツールをコンパイル時に変換することにより，VM イントロスペクションを行えるようにする．LLView を用いて作成された proc ファイルシステムを用いることで，既存の監視ツールを VM 内のシステムの監視に用いることもできる．

7. まとめ

本稿では，eBPF プログラムを動的に VM 内の OS に送り込み，VM 内の情報を安全かつ透過的に取得するシステム TeleBPF を提案した．eBPF アプリケーションが eBPF 関連システムコールを実行すると TeleBPF 共有ライブラリがそれを横取りし，VM 専用の通信機構を用いて VM 内の TeleBPF プロキシに転送する．そして，TeleBPF プロキシが代わりにシステムコールの実行を行い，その返り値を eBPF アプリケーションに返送する．eBPF アプリケーションが VM のメモリに直接アクセスすることにより，VM 内のリングバッファから高速に情報を取得することができる．実験より，TeleBPF を用いると VM 内で実行するよりも Sysmon が高速に動作することが分かった．

今後の課題は，より多くの BPF 関連システムコールや特殊ファイルへのアクセスに対応し，様々な既存の eBPF アプリケーションを実行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．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] Amazon Web Services, Inc.：Amazon CloudWatch, https://aws.amazon.com/cloudwatch/ (Accessed on 01/26/2024).
[2] IBM Corporation：IBM Cloud Monitoring, https://www.ibm.com/docs/ja/capm?topic=environment-configuring-cloud-monitoring (Accessed on 02/04/2024).
[3] T. Garfinkel and M. Rosenblum：A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proc. Network and Distributed Systems Security Symp, 191-206*, (2003).
[4] Advanced Micro Devices, Inc.：Secure Encrypted Virtualization API Version 0.24, (2020).
[5] Yasukata, K., Tazaki, H., Aublin, P. L., Ishiguro, K.：zpoline: a system call hook mechanism based on binary rewriting, *In 2023 USENIX Annual Technical Conference (USENIX ATC 23) (pp. 293-300)* (2023).
[6] IO Visor Project：BPF Compiler Collection (BCC), https://github.com/iovisor/bcc (Accessed on 01/26/2024).
[7] MarioHewardt：Sysinternals/SysmonForLinux, https://github.com/Sysinternals/SysmonForLinux (Accessed on 01/28/2024).
[8] Google Developers：Protocol Buffers, https://developers.google.com/protocol-buffers (Accessed on 01/26/2024).
[9] The kernel development community.：Syscall User Dispatch, https://docs.kernel.org/admin-guide/syscall-user-dispatch.html (Accessed on 01/26/2024).
[10] D. Benson：Protocol Buffers implementation in C, https://github.com/protobuf-c/protobuf-c (Accessed on 02/10/2024).
[11] Michael Kerrisk：vsock(7) - Linux manual page, https://man7.org/linux/man-pages/man7/vsock.7.html (Accessed on 02/03/2024).
[12] Chantra and Anakryiko：Automated upstream mirror for libbpf stand-alone build, https://github.com/libbpf/libbpf (Accessed on 02/21/2024).
[13] J. Fernandes：BPFd (Berkeley Packet Filter Daemon), https://github.com/joelagnel/bpfd (Accessed on 01/26/2024).
[14] Sysdig, Inc.：Sysdig: Security Tools for Containers, Kubernetes, and Cloud, https://sysdig.com/ (Accessed on 02/10/2022).
[15] 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* 論文集, *524-531* (2022).
[16] Y. Fu and Z. Lin：Space Traveling across VM: Automatically Bridging the Semantic Gap in Virtual Machine Introspection via Online Kernel Data Redirection, *Proc. Symp. Security and Privacy, 586–600* (2012).
[17] Y. Ozaki, S. Kanamoto, H. Yamamoto, and K. Kourai：Detecting System Failures with GPUs and LLVM, *In Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys 2019), pages 47-53*, (2019).