eBPFプログラムを送り込むことによるVM内のシステム監視

堀 恭介[1]　光来 健一[1]

概要：Amazon EC2 などの IaaS 型クラウドは仮想マシン（VM）内で動作するエージェントを用いて VM 内の情報を取得することが多い．この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行う必要があることであり，この作業を怠ると脆弱性となる可能性がある．それに対して，イントロスペクション方式はクラウド側が 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，開発が難しく，AMD SEV を用いてメモリが暗号化された VM には適用できない．本稿では，クラウド側から VM に eBPF プログラムを動的に送り込み，VM 内のシステム情報を安全に取得するシステム 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，eBPF プログラムの実行時に検査が行われるため OS 内で安全に実行することができる．TeleBPF は BPF 関連システムコールをフックし，VM 内の TeleBPF プロキシに透過的に転送する．TeleBPF プロキシは転送されたシステムコールを代理実行し，その結果をクラウド側に返す．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，システムコールを転送するオーバヘッドを測定した．

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，ユーザが VM 内のシステムを自由に管理することが可能である．クラウド側は VM 内のシステムの負荷や状態を監視することにより，VM のオートスケールやセキュリティ向上に活用することができる．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，クラウド側はエージェントと通信して情報を取得する．例えば，Amazon CloudWatch エージェント [1] は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．

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．この問題を解決するために，イントロスペクション方式 [2] が用いられている．この方式は VM の外部のクラウド側から VM のメモリや仮想ディスクなどに直接アクセスし，OS のデータ構造やファイルシステムを解析することで情報を取得する．しかし，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，ユーザが管理している VM 内のシステムに適用するのは容易ではない．また，最近の CPU によって提供されている AMD SEV [3] などによってメモリが暗号化された VM に対しては利用することができない．

本稿では，クラウド側から VM に eBPF プログラムを動的に送り込んで実行し，VM 内のシステムを監視するシステム 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，システム内のイベント発生時に性能など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 プログラムにアクセスして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よる BPF 関連システムコールの呼び出しをフックして VM に転送する．具体的には，eBPF プログラムのロードやマップの処理を行う eBPF システムコールと，eBPF プログラムをイベントに関連づけるために用いられる eBPF イベント制御システムコールを転送する．転送されたシステムコールは VM

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

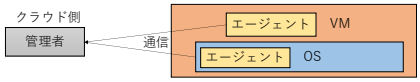

図 1 エージェント方式

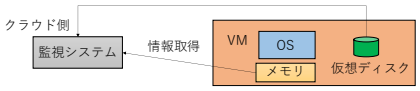

図 2 イントロスペクション方式

内で動作する TeleBPF プロキシによって VM 内の OS に対して実行される．転送の際には，システムコールの引数をシリアライズすることでバイト列に変換し，プロキシで元のデータ構造に復元する．クラウド側で BPF Compiler Collection (BCC) [4] を用いて作成した eBPF アプリケーションを実行し，TeleBPF を用いて VM 内のシステムの情報が取得できることを実験により確認した．また，eBPF アプリケーションの実行時間を比較して TeleBPF のオーバヘッドを調べた．

以下，2 章では VM からの従来の情報取得方法である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方法である TeleBPF を提案する．4 章ではシステムコールのフックおよびプロキシを用いた TeleBPF の実装について説明する．5 章では TeleBPF の性能を調べる実験を行い，TeleBPF の動作確認とオーバヘッドについて述べる．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，クラウドは VM の性能を監視することでオートスケールを行っている．その際に，VM の仮想 CPU などの情報だけでなく，VM 内のシステムの情報も利用することで，より的確にスケールアウトやスケールインを行うことができる．また，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もできる．クラウドによる情報取得方法としてエージェント方式とイントロスペクション方式が主に用いられている．

エージェント方式は**図 1** 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，クラウド側はエージェントと通信して情報を取得する方式である．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．このようなエージェントはクラウドによって用意されているパッケージを用いて容易にインストールすることができる．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 [5] 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．このように，カーネル内のエージェントはより詳細な情報を取得することができる．

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．もし，このような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する．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，システムが不安定になり信頼性が低下する可能性がある．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正常に動作しなくなる可能性がある．

一方，イントロスペクション方式 [2] は**図 2** のようにクラウド側から VM のメモリや仮想ディスクなどに直接アクセスし，VM 内のシステムの情報を取得する方式である．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で，システム性能に関する情報を取得したり，プロセスやネットワークなどの情報から侵入を検知したりすることができる．また，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で，設定ファイルやログファイルなどを取得することもできる．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解析は低レベルであるため，監視システムの開発が難しくなる．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 [3] などの CPU のセキュリティ機能を利用している VM については，クラウド側からメモリ上の情報を取得することができない．SEV は VM のメモリを CPU 内部の暗号鍵で暗号化するため，

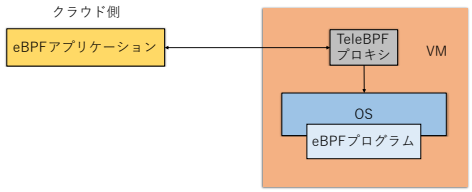

図 3 TeleBPF

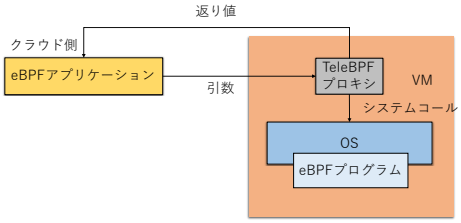

図 4 eBPF 関連システムコールの転送

VM の外部からはアクセスすることができないためである．仮想ディスクについても，VM 内のシステムによって暗号化されているとクラウド側からはアクセスすることができない．

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て，より複雑なプログラムを実行できるようにした Linux の機能である．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 3** 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．このように，TeleBPF においては eBPF プログラムがエージェントの役割を果たす．

従来のエージェント方式と比べた TeleBPF の利点として，eBPF プログラムを VM 内に動的に送り込むことができることが挙げられる．eBPF アプリケーションを実行するたびに必要な eBPF プログラムが送り込まれるため，VM のユーザが事前にエージェントをインストールしておく必要はない．また，常に最新の eBPF プログラムを送り込むことができるため，VM のユーザがエージェントの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，OS 内の詳細な情報を取得することができ，プロセスとして実行されるエージェントに比べて取得できない情報は少なくなる．また，eBPF プログラムはロード時に検査器を用いて検査され，プログラム外へのジャンプや無限ループなどの安全でない処理が実行されないことが保証される．そのため，カーネルモジュールとして実行されるエージェントとは異なり，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比べた利点として，SEV などによる VM のメモリ暗号化の影響を受けずに情報を取得できることが挙げられる．送り込まれた eBPF プログラムは VM 内で実行されるため，CPU によって復号されたメモリ上のデータにアクセスすることができる．また，OS レベルのデータ構造や関数を用いて情報を取得することができるため，低レベルな VM のメモリ解析が不要になる．eBPF において OS のデータ構造はある程度，抽象化されているため，OS バージョンの違いの影響も受けにくい．

eBPF アプリケーションが eBPF プログラムを透過的に VM 内に送り込んでアクセスできるようにするために，TeleBPF では eBPF システムコールの転送を行う．eBPF システムコールは eBPF プログラムの OS 内へのロードやデータを共有するためのマップの作成や読み書きなどを行うために，eBPF アプリケーションによって用いられる．TeleBPF は**図 4** のように，eBPF アプリケーションが実行した eBPF システムコールをフックし，その引数をシリアライズして VM 内の TeleBPF プロキシに転送する．例えば，eBPF プログラムをロードする際にはその名前や種類，バイトコードなどが eBPF システムコールの引数で指定される．TeleBPF プロキシは引数をデシリアライズして eBPF システムコールを実行し，その返り値をクラウド側に転送して eBPF アプリケーションの実行に利用する．

TeleBPF は eBPF アプリケーションが実行する eBPF イベント制御システムコールについてもフックして転送を行う．eBPF イベント制御システムコールはシステム内のイベントに eBPF プログラムを関連づけ，イベントが発生した時に指定した eBPF プログラムが実行されるようにするために用いられる．例えば，OS 内の指定された関数が実行された時に eBPF プログラムを呼び出すようにすることができる．eBPF システムコールの場合と同様に，TeleBPF は eBPF イベント制御システムコールの種類と引数を TeleBPF プロキシに転送してシステムコールを実行し，その返り値を eBPF アプリケーションに転送する．

4. 実装

KVM を用いて作成された VM 上で動作する Linux 5.4

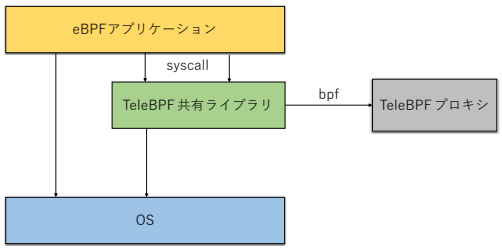

図 5 システムコールのフック

に対して TeleBPF を実装した．

4.1 システムコールのフック

TeleBPF は LD_PRELOAD 環境変数を用いて**図 5** のように実行時にシステムコール関数を置き換えることで，eBPF アプリケーションが実行するシステムコールのフックを行う．そのために，置き換えるシステムコール関数を定義した TeleBPF 共有ライブラリを提供する．しかし，すべてのシステムコールに対応するシステムコール関数があるとは限らず，TeleBPF がフックする必要がある bpf システムコールや perf_event_open システムコールにはシステムコール関数が用意されていない．これらのシステムコールは syscall 関数を用いて実行されるため，TeleBPF 共有ライブラリでは syscall 関数を定義する．syscall 関数はシステムコール番号と任意の引数をとるため，システムコール番号に応じて引数を取り出す．

4.2 bpf システムコールの転送

eBPF アプリケーションが bpf システムコールを実行すると，TeleBPF 共有ライブラリの中で定義した syscall 関数が呼び出される．引数で指定されたシステムコール番号がこのシステムコールのものであった場合，TeleBPF は syscall 関数の可変引数から eBPF コマンド番号，bpf_attr 共用体のアドレス，共用体のサイズを取り出す．eBPF コマンド番号は eBPF プログラムのロードやマップの作成などの機能にそれぞれ割り当てられた番号である．bpf_attr 共用体にはそれぞれの eBPF コマンドが必要とするデータが格納されている．TeleBPF は VM 内の TeleBPF プロキシと通信して bpf システムコールのこれらの引数を転送するが，bpf_attr 共用体については eBPF コマンドに応じて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，syscall 関数の返り値とする．

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，ライセンスなどの情報を転送する．プログラム名やライセンスは文字列，バイトコードはバイト列であるため，必要なサイズ分のデータを転送する．TeleBPF プロキシはシステムコールの返り値として，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．また，マップから値を取得する eBPF コマンドの場合，bpf_attr 共用体に格納されているマップにアクセスするためのファイル記述子，取得するデータのキーなどを転送する．キーのサイズはマップを作成する eBPF コマンドが実行される際に指定されるため，そのサイズ分のデータを転送する．TeleBPF プロキシがシステムコールを実行すると bpf_attr 共用体に取得した値が格納されるため，マップ作成時に指定されたサイズ分の値のデータを TeleBPF 共有ライブラリに返す．

4.3 perf_event_open システムコールの転送

perf_event_open システムコールは性能監視のためのイベントを設定するために用いられる．eBPF アプリケーションがこのシステムコールを実行すると，TeleBPF 共有ライブラリの中で定義した syscall 関数が呼び出される．指定されたシステムコール番号がこのシステムコールのものであった場合，TeleBPF は syscall 関数の可変引数から perf_event_attr 構造体，監視対象のプロセス ID，CPU 番号などを取り出す．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．TeleBPF 共有ライブラリはこれらの引数を VM 内の TeleBPF プロキシに転送し，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．このファイル記述子は ioctl システムコールに指定して用いられるため，一般のファイル記述子と区別するために，TeleBPF 共有ライブラリで固定値を加えて syscall 関数の返り値とする．

4.4 ioctl システムコールの転送

eBPF アプリケーションが ioctl システムコールを実行すると，TeleBPF 共有ライブラリの中で定義した ioctl 関数が呼び出される．ioctl システムコールは様々な用途に利用されるため，第 1 引数で指定されたファイル記述子の値が perf_event_open システムコールの返り値に固定値を加えた値の範囲内であれば，固定値を減じた上で TeleBPF プロキシへの転送を行う．それ以外の場合には従来の ioctl 関数を呼び出す．転送を行う場合，第 2 引数で指定されたリクエスト番号に応じて eBPF プログラムの関連づけや eBPF プログラムの有効化・無効化のためのコマンドを TeleBPF プロキシに送信する．ioctl 関数は第 3 引数として任意の型のデータを取るため，指定されたコマンドに応じて適切な型を指定してデータを取り出して転送する．

```
message BpfProgLoad{
  int32 prog_type = 1;
  :
  bytes insns = 3;
  string license = 4;
  :
}
```

図 6 .proto ファイルの例

表 1 実験で用いたホストと VM

	ホスト	VM
CPU	Intel Core i7-10700	1
メモリ	64GB	4GB
OS	Linux 5.4	Linux 5.4
BCC	0.22.0	-
プロトコルバッファ	3.6.1	3.19.1
仮想化ソフトウェア	4.2.1	-

4.5 プロトコルバッファを用いたデータ転送

TeleBPF では bpf_attr 共用体や perf_event_attr 構造体などの大量のメンバのデータをプロキシに転送する必要がある．これらのデータの転送を容易かつ効率よく行えるようにするために，プロトコルバッファ [6] を用いてシリアライズして転送する．TeleBPF は C 言語で実装しているため，プロトコルバッファの C 言語用の実装 [7] を用いた．プロトコルバッファは.proto ファイルに転送するデータを定義する．**図 6** に eBPF プログラムをロードする際に使われる.proto ファイルの一部を示す．各フィールドの右の数値はフィールドを識別するための番号である．int32 はそのフィールドの型であり，例えば string は文字列，bytes はバイト配列を表している．TeleBPF 共有ライブラリでは，フィールドが文字列の場合はポインタを格納し，バイト配列の場合はポインタとサイズを格納する．それをシリアライズしてバイト列にして転送し，TeleBPF プロキシでデシリアライズして元のデータ構造を復元する．プロキシでは，文字列もバイト配列もポインタを使ってアクセスすることができる．

5. 実験

BPF Compiler Collection (BCC) を用いて eBPF アプリケーションを作成し，TeleBPF の有効性を確認する実験を行った．まず，TeleBPF を用いて eBPF プログラムを VM に送り込み，VM 内の情報が取得できることを確認した．次に，eBPF アプリケーションの実行時間を測定した．比較として，TeleBPF を用いずに eBPF アプリケーションをホストだけで実行した場合についても測定を行った．また，BCC を改変して TeleBPF を実装した場合についても測定した．この実験に使用したホストと VM を**表 1** に示す．

```
hori@hori-desktop:~/telebpf$ sudo LD_PRELOAD=./telebpf.so python3 readkaisuu.py
counter=6
counter=12
counter=15
counter=18
counter=22
```

図 7 read システムコールの実行回数の出力

表 2 イベントの対応状況

イベント	動作
kprobes	○
kretprobes	○
Tracepoints	○
uprobes	○
uretprobes	○
USDT probes	○
Raw Tracepoints	×
system call tracepoints	○
kfuncs	カーネルが未対応
kretfuncs	カーネルが未対応
LSM Probes	カーネルが未対応
BPF ITERATORS	カーネルが未対応

5.1 動作確認

TeleBPF を用いて eBPF アプリケーションが想定した動作を行うかどうかを確認する実験を行った．まず，指定したシステムコールを実行した時にログを出力する eBPF アプリケーションを実行した．実験の結果，eBPF プログラムを VM 内に送り込んだ後，VM 内でプログラムを実行するたびにログにメッセージが出力されることを確認した．次に，read システムコールが実行されるたびに実行回数をマップに記録する eBPF プログラムを作成し，1 秒ごとにマップから実行回数を取得して表示する eBPF アプリケーションを実行した．その結果，**図 7** のようになり，VM 内でファイルやネットワークからの読み込みを行うと実行回数が増えることが確認できた．

様々なイベントに eBPF プログラムを関連づける eBPF アプリケーションを作成し，TeleBPF を用いて実行した結果を**表 2** に示す．Raw Tracepoints は eBPF コマンドの転送がまだ実装できていないため動作しなかった．kfuncs 以降のイベントは用いたカーネルがサポートしていないため，動作が確認できなかった．

5.2 オーバヘッド

TeleBPF を用いて read システムコールの実行回数を取得する eBPF アプリケーションを実行し，実行時間を測定した．測定は計 5 回行い，その平均を用いた．実験の結果，eBPF プログラムをロードするのにかかる時間は**図 8** のようになった．システムコールの転送を行わない場合と比べて，TeleBPF では 100ms 程度長い時間がかかることが分かった．しかし，eBPF プログラムのロードは eBPF アプリケーションの実行開始時にのみ行われるのが一般的であるため，影響が大きいオーバヘッドではないと考えられる．

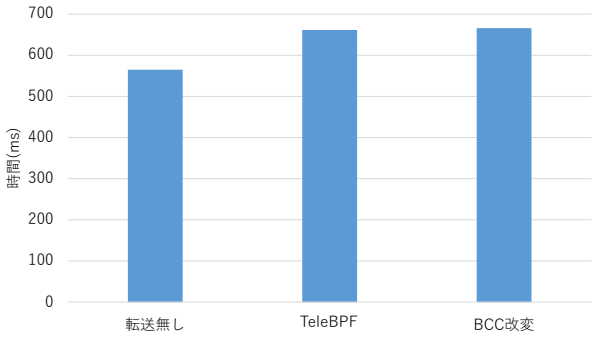

図 8 eBPF プログラムのロードにかかる時間

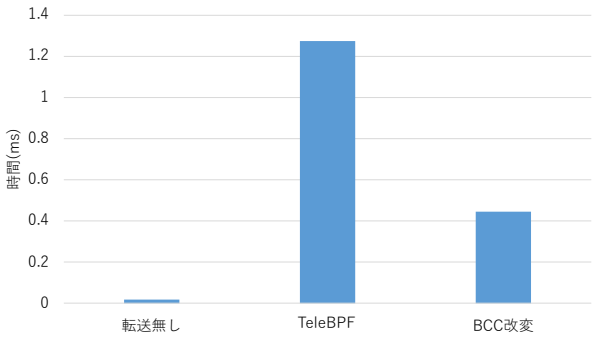

図 9 マップからの情報取得にかかる時間

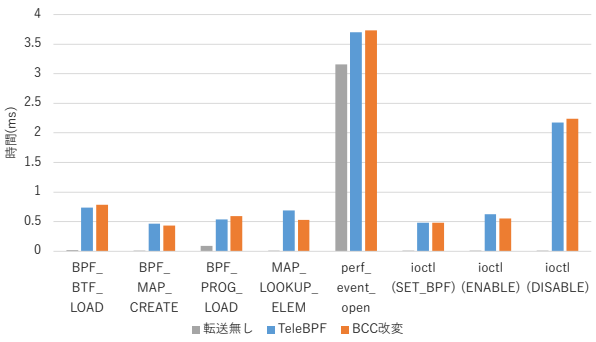

図 10 システムコールの実行時間の比較

一方，マップを用いた情報取得にかかる時間は**図 9** のようになった．システムコールの転送を行わない場合と比べると 1.26ms 増加しているが，1 秒に 1 回実行されるだけであれば問題はないと考えられる．BCC を改変した実装と比べてもオーバヘッドが大きいが，この原因については現在調査中である．

次に，オーバヘッドの内訳を調べるために，この eBPF アプリケーションが実行したシステムコールのそれぞれの実行時間を測定した．その結果は**図 10** のようになった．システムコールを転送するオーバヘッドは 0.71〜2.18ms であることが分かった．ioctl システムコールでイベントと eBPF プログラムの関連付けをやめる際のオーバヘッドだけが特に大きかった．しかし，この処理は eBPF アプリケーションの終了時にのみ行われるため，問題になることはないと考えられる．

6. 関連研究

BPFd [8] は 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れていたが，現在は開発が終了している．eBPF プログラムのロードや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 の方が汎用性が高い．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 [9] は従来は 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 [10] 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，ネストした仮想化を用いてハイパーバイザ内で動作させる手法，OS の管理外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 [11] 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 [12] 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 内の情報を安全に取得するシステム TeleBPF を提案した．TeleBPF は eBPF アプリケーションによる eBPF 関連システムコールの呼び出しをフックし，そのシステムコールを 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，その返り値をクラウド側の eBPF アプリケーションに転送する．これにより，既存の eBPF アプリケーションを用いて VM 内の詳細な情報を取得することができる．実験結果より，VM にシステムコールを転送するオーバヘッドは大きいが，1 回の情報取得にかかる時間は 1.3ms であることが分かった．

今後の課題は，より多くの BPF 関連システムコールやシステムファイルに対応し，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．また，TeleBPF プロキシにアクセスする際のセキュリティの強化も課題である．現在の実装では，クラウド側から不特定多数が TeleBPF プロキシに接続し，VM の情報を自由に入手できてしまう．そのため，認証や通信暗号化などを行うことによりアクセスを制限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．

参考文献

[1] Amazon Web Services, Inc.：Amazon CloudWatch, `https://aws.amazon.com/cloudwatch/` (Accessed on 02/14/2022).
[2] T. Garfinkel and M. Rosenblum：A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proc. Network and Distributed Systems Security Symp, 191-206*, (2003).
[3] Advanced Micro Devices, Inc.：Secure Encrypted Virtualization API Version 0.24, (2020).
[4] IO Visor Project：BPF Compiler Collection (BCC), `https://github.com/iovisor/bcc` (Accessed on 01/31/2022).
[5] IBM Corporation：IBM Cloud Monitoring, `https://cloud.ibm.com/docs/monitoring` (Accessed on 06/20/2022).
[6] Google Developers：Protocol Buffers, `https://developers.google.com/protocol-buffers` (Accessed on 02/02/2022).
[7] D. Benson：Protocol Buffers implementation in C, `https://github.com/protobuf-c/protobuf-c` (Accessed on 06/19/2022).
[8] J. Fernandes：BPFd (Berkeley Packet Filter Daemon), `https://github.com/joelagnel/bpfd` (Accessed on 02/09/2022).
[9] Sysdig, Inc.：Sysdig: Security Tools for Containers, Kubernetes, and Cloud, `https://sysdig.com/` (Accessed on 06/20/2022).
[10] 能野智玄, 光来健一：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，研究報告システムソフトウェアとオペレーティング・システム *(OS), 1-8* (2021.07.13).
[11] 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).
[12] 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).