クラウドにおいてVM内情報を利用可能なP4プログラムの安全な実行

岩井 正輝[1]　光来 健一[1]

概要：最近，用いられるようになってきているプログラム可能なネットワークスイッチは，P4 言語などのソフトウェアによってパケット転送を制御できる．仮想マシン (VM) 向けには仮想 P4 スイッチが開発されており，仮想スイッチの内部で P4 プログラムを実行する．ユーザが独自の P4 プログラムを仮想 P4 スイッチにロードできれば，VM 内部の情報を使用したきめ細かいパケット転送が可能となる．しかし，クラウドにおいて提供される仮想 P4 スイッチは信頼できるとは限らず，仮想 P4 スイッチもユーザの P4 プログラムを信頼できない．そこで本稿では，ユーザごとに用意された P4 VM を用いて，各ユーザの P4 プログラムをクラウドの仮想スイッチの外部で安全に実行する P4Shield を提案する．P4Shield は P4 VM を機密 VM として実行することで，クラウドから P4 プログラムを保護する．また，P4 VM 内で作成する uBPF サンドボックスにユーザの P4 プログラムを閉じ込めることで，クラウドの仮想スイッチをユーザの P4 プログラムから保護する．P4 プログラムがユーザの VM 内情報を利用できるようにするために，P4 VM とユーザ VM 間の共有メモリに格納された情報にアクセスするための P4 外部関数を提供する．P4Shield を Open vSwitch を用いて実装し，その有効性を確認した．

1. はじめに

近年，ネットワークスイッチにおいて P4 言語 [4] を用いてパケットの転送処理をプログラムできるようになってきている．例えば，高度なパケットフィルタリングを行ったり，パケットのデータ書き換えを行ったりすることができる．P4 により，対応するネットワークスイッチの登場を待つことなく，新たな技術を利用することが可能となる．クラウドにおいて仮想マシン (VM) をネットワークに接続する際には仮想スイッチが用いられるが，P4 に対応した仮想スイッチも開発されている [17].

P4 プログラムは管理者によって仮想 P4 スイッチにロードされるが，ユーザが独自の P4 プログラムをロードできるようになれば，VM ごとに柔軟なパケット転送処理が実現できるようになる．加えて，P4 プログラムの中で VM 内の情報も利用できれば，よりきめ細かいパケット処理が可能となる．しかし，クラウドによって提供されている仮想スイッチは信頼できるとは限らないため，ユーザの P4 プログラムを改ざんされたり，P4 プログラムが取得した VM 内情報を盗聴されたりするリスクがある．逆に，ユーザの P4 プログラムの挙動が仮想スイッチに影響を与える恐れもある．

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

そこで本稿では，VM 内情報を利用できるように拡張した P4 プログラムをユーザごとに用意される VM (P4 VM) で安全に実行する P4Shield を提案する．P4Shield は P4 VM を機密 VM として実行することで，クラウドによる P4 プログラムへの攻撃を防ぐ．機密 VM のメモリやレジスタは AMD SEV [6] や Intel TDX [12] などの Trusted Execution Environment (TEE) を用いてプロセッサにより暗号化され，完全性が検証される．また，P4Shield は P4 プログラムを P4 VM 内に作成された uBPF サンドボックス [5] に隔離して実行することにより，P4 プログラムの異常な動作からクラウドの仮想スイッチを保護する．uBPF は，ロード時の検証機能を有する安全なプログラム実行のためのフレームワークである．

P4Shield では，仮想スイッチが受信したパケットを適切な P4 VM に転送して P4 プログラムを実行し，実行結果に基づいてパケットの転送処理を行う．このとき，P4Shield は P4 プログラムがユーザの VM 内の情報を利用できるようにする．P4 プログラムは VM 内情報に直接アクセスできないため，P4Shield は VM 内情報を取得するための外部関数を提供する．P4 における外部関数は，P4 プログラムの外部で定義されたプログラムを実行するための機構である．P4 プログラムからユーザ VM 内の情報を安全に取得できるようにするために，ユーザ VM は P4 VM との間

で確立した共有メモリに暗号化した情報を格納する．そして，P4 プログラムがユーザ VM 内の情報を取得するための API を外部関数として定義し，外部関数の中で uBPF ランタイムに実装したヘルパー関数を実行することで共有メモリにアクセスする．

P4 Shield を Open vSwitch [3]，uBPF ランタイム [2]，AMD SEV による機密 VM を使用して実装した．P4 Shield を用いて，P4 VM 内の P4 プログラムがユーザ VM 内の情報を利用してパケットフィルタリングを行えることを確認する実験を行った．その結果，ユーザ VM 内で使用されている TCP メモリがあらかじめ指定した閾値を超えたときに，P4 Shield が新たな TCP 接続を拒否できることを確認した．また，P4 Shield における TCP および UDP のレイテンシとスループットを測定し，性能低下は許容範囲内であることを確認した．

以下，2 章でユーザがクラウドの仮想スイッチに P4 プログラムをロードできるようにするための課題を示す．3 章で VM 内情報を利用できるように拡張した P4 プログラムをユーザごとに用意される P4 VM で安全に実行する P4 Shield を提案する．4 章で P4 Shield の実装について説明し，5 章で P4 Shield の動作確認と性能測定のために行った実験について述べる．6 章で関連研究を示し，7 章で本稿をまとめる．

2. ユーザによる仮想 P4 スイッチの利用

P4 言語は，ネットワークデータプレーンのプログラミングに使用されるドメイン固有言語である．プログラマが高レベルで宣言的にデータプレーンの動作を定義できる．P4 プログラムはパケットデータを入力として受け取り，パーサブロックでパケットヘッダを解析する．次に，コントロールブロックでパケットフィルタリングやパケットヘッダの書き換えなどのパケット処理のパイプラインを実行する．最後に，デパーサブロックで変更されたパケットヘッダからパケットデータを再構築する．このような P4 プログラムは，P4 ネットワークスイッチで実行される．

物理的な P4 スイッチに加えて，VM 向けにも P4rt-OVS [17] や IPDK Networking Recipe [16] などの P4 対応の仮想スイッチが開発されている．仮想環境では，ハイパーバイザ上で VM を実行する各ホスト内に仮想スイッチが作成され，各 VM の仮想 NIC を接続することで仮想ネットワークを構築できる．さらに，仮想スイッチをホストの物理 NIC に接続することにより他のホストの内部の仮想スイッチと接続することができ，リモートの VM や物理ホストとの通信が可能になる．仮想 P4 スイッチは，**図 1** に示すように，パケットの転送時に P4 プログラムを実行する．

P4 プログラムは仮想インフラの一部である仮想スイッチによって処理されるため，通常，ネットワークやホストの

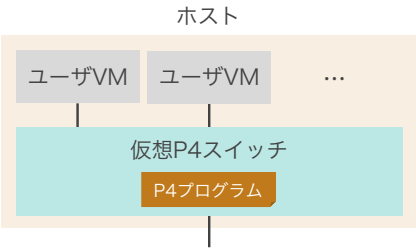

図 1: 仮想 P4 スイッチ

管理者のみがロードすることができる．例えば，P4rt-OVS では ovs-ofctl コマンドを用いて P4 プログラムがロードされる．VM のユーザが独自の P4 プログラムを仮想スイッチにロードできるようになれば，VM ごとにより柔軟なパケット転送処理が実現できるようになる．さらに，VM と連携してその内部の情報も利用することができれば，よりきめ細かいパケット処理が可能となる．例えば，パケットを送受信するアプリケーションのプロセス名や当該プロセスの実行ユーザ情報を利用したパケットフィルタ [19] などを実現できる．

しかし，ユーザが作成した P4 プログラムを仮想 P4 スイッチで実行できるようにするには大きく 3 つの課題がある．第 1 に，クラウドで提供される仮想 P4 スイッチはユーザにとって信頼できるとは限らない．信頼できないクラウド管理者によって管理されている場合，ユーザがロードした P4 プログラムに対して様々な攻撃が考えられる．例えば，P4 プログラム自体や P4 プログラムが参照する VM 内情報を盗聴されると，P4 プログラムやユーザ VM の機密性が失われる危険性がある．意図しないパケット転送を実行するように P4 プログラムを改ざんされると，P4 プログラムの完全性も侵害される．さらに，仮想 P4 スイッチを通るすべてのパケットに対して P4 プログラムを実行しないことによって，可用性が失われることもある．

第 2 に，ユーザがロードした P4 プログラムに意図的または意図せずに引き起こされる問題が含まれていると，仮想 P4 スイッチの挙動に大きな影響を与える恐れがある．例えば，P4 プログラムが CPU やメモリなどのリソースを大量に消費する場合，すべてのパケットの転送が遅延する可能性がある．P4 プログラムやそれを実行するランタイムに脆弱性があった場合，仮想 P4 スイッチの制御を奪われるリスクもある．実際に，P4 プログラムが無限ループに陥り，以降のパケットをすべて破棄したり，宛先ポートが正しく指定されていないパケットをすべて特定のポートに差し向けたり，特定条件下において以前に転送したパケットの情報を漏洩したりするなど，様々な脆弱性が指摘されている [7]．

第 3 に，P4 プログラムはパケットヘッダとペイロードに含まれる情報にしかアクセスできない．P4 プログラムが VM 内情報を取得するには，ユーザ VM から情報を取

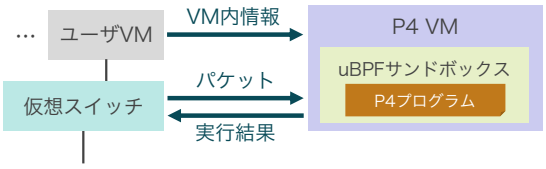

図 2: P4 Shield のシステム構成

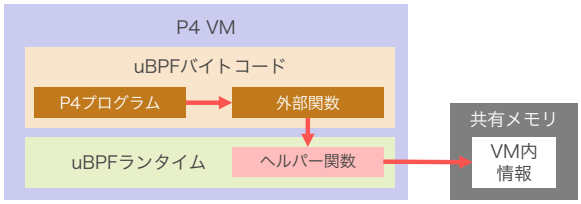

図 3: 外部関数を使用した VM 内情報の取得

得できるように P4 を拡張しなければならない．考えられる解決策の 1 つは，ユーザ VM とのネットワーク通信用の API を追加することである．しかし，パケット処理ごとに通信を行っていては，実用に耐え得る転送性能を得ることが難しくなる．このオーバヘッドを削減するには，VM イントロスペクション [8] を用いて VM のメモリ上の情報を直接，取得する方法が考えられる．この手法を用いれば高速化が可能になるが，ユーザ VM が機密 VM として実行されている場合には適用できない．VM のメモリがプロセッサによって暗号化されており，P4 プログラムがユーザ VM のメモリ内の情報にアクセスできないためである．

本稿における脅威モデルは以下の通りである．ユーザは，仮想スイッチと管理者を含むクラウドを信頼しない．しかし，TEE ハードウェアと機密 VM 内で実行されるソフトウェアは信頼でき，脆弱性がないことを仮定する．攻撃として，クラウドがユーザの P4 プログラムとユーザ VM から取得した情報を盗聴，改ざんする攻撃を考える．クラウドは，P4 プログラムがユーザ VM から情報を取得するための通信路に対する攻撃を行う恐れがあるほか，ユーザの P4 プログラムの実行をバイパスする可能性もある．これらに加えて，リソースの枯渇など，ユーザの P4 プログラムが仮想スイッチに悪影響を及ぼす攻撃も考える．

3. P4 Shield

P4 Shield は，ユーザごとに用意された専用の VM である P4 VM を用いて，クラウドの仮想スイッチが各ユーザの P4 プログラムを安全に実行できるようにする．仮想スイッチがユーザ VM に関連するパケットを受信すると，**図 2** に示すように，まず共有メモリを介して対応する P4 VM にパケットを転送する．次に，P4 VM は P4 プログラムを実行し，パケットの転送・破棄の判定や書き換えたパケットのデータを仮想スイッチに返送する．P4 プログラムが処理にユーザ VM 内の情報を必要とする場合，共有メモリを用いて必要な情報を取得する．P4 プログラムの実行結果に基づいて，仮想スイッチはパケットを転送または破棄する．

P4 Shield は P4 VM を機密 VM として実行することで，クラウドからユーザの P4 プログラムを保護する．機密 VM は，最近のプロセッサで提供されている AMD SEV や Intel TDX などの Trusted Execution Environment (TEE) によって保護される．P4 VM のメモリとレジスタの状態は暗号化されるため，クラウドは P4 プログラムや P4 プログラムによって取得された VM 内情報を盗聴することはできない．完全性も維持されるため，クラウドは P4 プログラムを改ざんできない．また，従来の仮想 P4 スイッチと異なり，P4 プログラムを仮想スイッチ内で実行しないようにすることにより P4 プログラムから仮想スイッチを保護することができる．P4 プログラムが異常な動作をしたとしても，その影響を P4 VM に閉じ込めることができる．

P4 Shield では 1 つの P4 VM の中で複数の P4 プログラムを実行できるため，uBPF サンドボックスを使用して P4 プログラム同士を隔離する．uBPF は eBPF [1] のユーザ空間実装であり，OS カーネルにロードされたプログラムを安全に実行するために Linux や Windows で使用されているフレームワークである．P4 Shield はユーザが作成した P4 プログラムを uBPF バイトコードにコンパイルし，P4 VM で実行される uBPF ランタイムにロードする．このとき，uBPF ランタイムは無限ループや安全でない操作などが含まれていないかの検証を行う．併せて，Just-In-Time (JIT) コンパイルを使用してバイトコードをネイティブコードにコンパイルし，効率的に実行する．各 uBPF バイトコードには個別の実行環境が提供されるため，互いに影響を及ぼしたり，仮想スイッチに影響を及ぼすことはない．

P4 プログラムからユーザ VM 内の情報を取得できるようにするために，P4 Shield では P4 VM と各ユーザ VM 間の共有メモリを使用する．共有メモリを介した通信はネットワーク通信よりも効率的であるが，P4 Shield はさらに通信の同期も排除する．ユーザ VM は，P4 プログラムが必要とするパケットに関連する情報を定期的に共有メモリに格納する．P4 VM 内の P4 プログラムは，その情報を共有メモリからすぐに取得できる．ただし，TEE によって暗号化されたメモリは VM 間で共有できないため，共有メモリは TEE による暗号化の対象外とする必要がある．そのため，クラウドは共有メモリ内の情報を盗聴・改ざんできる．このような攻撃を防ぐためにユーザ VM は情報を暗号化し，メッセージ認証コード (MAC) とともに共有メモリへ格納する．P4 VM は共有メモリ内の情報を復号し，整合性検査を行う．そのために必要な暗号鍵はユーザ VM と P4 VM 間で事前に安全に共有する．

P4 Shield は**図 3** に示すように専用の外部関数を提供することにより，P4 プログラムが共有メモリ上の VM 内情報

にアクセスできるようにする．外部関数は P4 言語の仕様に含まれているため，言語仕様を拡張する必要はない．P4 外部関数は uBPF バイトコードにコンパイルされ，同じく uBPF バイトコードにコンパイルされた P4 プログラムにリンクされる．そのため，外部関数も uBPF サンドボックス内で実行され，VM 内情報が格納されている共有メモリに直接アクセスすることができない．そこで，P4 外部関数は P4Shield 用に拡張された uBPF ランタイムによって提供されるヘルパー関数を呼び出す．このヘルパー関数が共有メモリにアクセスし，指定された VM 内情報を P4 外部関数を経由して P4 プログラムに返却する．

P4 プログラムが正常に実行されたかどうかをユーザが確認できるようにするために，P4 VM は P4 プログラムの実行ログをユーザ VM との間の共有メモリに格納する．クラウドの仮想スイッチは，P4 VM にパケットを転送せずにパケット処理を行うことで P4 プログラムの実行をバイパスすることが可能である．このような攻撃を検知するために，ユーザ VM は共有メモリに格納された実行ログを定期的に取得し，パケット送受信の統計データと照合する．P4 プログラムの実行回数が統計データに示されるユーザ VM での処理パケット数を下回った場合，P4 プログラムが正常に実行されていないことを検知することができる．クラウドによる共有メモリ上の実行ログの改ざんを防ぐために，P4 VM は実行ログから算出された MAC を共有メモリに格納し，ユーザ VM で MAC を再計算して比較することで改ざんを検出する．

4. 実装

P4Shield を Open vSwitch (OVS) 3.2.1 [3]，Data Plane Development Kit (DPDK) 22.11.7 [11]，uBPF ランタイム [5]，KVM 上で動作する AMD SEV を使用した機密 VM を用いて実装した．

4.1 OVS の拡張

P4Shield は OVS のデータパスを拡張してパケットデータを P4 VM に送信し，P4 プログラムの実行結果を受信する．データパスとは，ネットワークパケットの処理および転送を行う転送プレーンのことである．OVS はユーザ空間データパス，カーネルデータパス，DPDK データパスなどのいくつかのデータパスを提供しているが，今回ユーザ空間データパスと DPDK データパスに P4Shield を実装した．ユーザ空間データパスは効率が低いため，本稿では DPDK データパスでの実装について示す．DPDK を用いることで，**図 4** に示すように OVS が OS カーネルをバイパスし，ユーザ空間のみでパケットを処理できるようになる．これにより，割り込みやシステムコールのオーバヘッドを削減し，効率的なパケット処理を実現できる．DPDK は Poll Mode Driver (PMD) を用いて必要な数の CPU コアを占有し，物理 NIC および仮想 NIC の送信 (TX) キューと受信 (RX) キューをポーリングする．

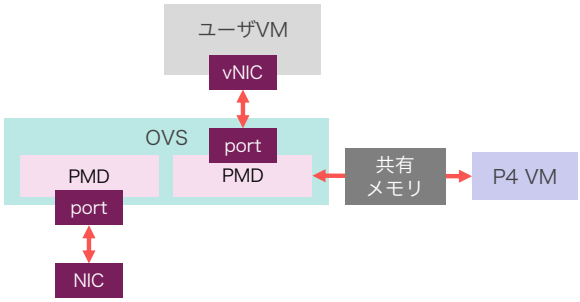

図 4: OVS における DPDK データパスの拡張

4.2 uBPF ランタイムの拡張

uBPF バイトコードを実行するための uBPF ランタイムを拡張し，OVS と連携して VM 内情報を利用する P4 プログラムを実行できるようにした．P4Shield はユーザの P4 プログラムから生成された uBPF バイトコードを，P4 VM で実行される uBPF ランタイムに事前にロードする．その際に uBPF バイトコードを JIT コンパイルし，ネイティブ実行できるようにする．初期化処理後，uBPF ランタイムは OVS と P4 VM 間の共有メモリをポーリングし，OVS が共有メモリにパケットデータを格納するまで待機する．OVS からパケットデータを受信すると，パケットデータを uBPF バイトコードの引数にして実行を開始する．実行を終えると，uBPF ランタイムは結果を共有メモリに格納し OVS に返却する．

uBPF ランタイムは，uBPF バイトコードに新たなヘルパー関数である `read_vm_info()` を提供する．このヘルパー関数は，P4 VM とユーザ VM との間に確立された共有メモリに格納されている VM 内情報にアクセスする．VM 内情報はユーザ VM によって AES-GCM で暗号化されているため，ヘルパー関数はそれを復号し，認証タグを検証する．ヘルパー関数は共有メモリの先頭からのオフセットを引数として受け取り，そのメモリ領域に格納されている値を返す．uBPF ランタイムは，OVS から渡されたパケットヘッダの送信元または宛先 IP アドレスによって，アクセスするユーザ VM を識別する．P4Shield は，共有メモリの各オフセットに格納する VM 内情報を定義している．uBPF ランタイムは様々なヘルパー関数に一意の番号を割り当て，番号と関数ポインタ間のマッピングを管理している．uBPF バイトコードがオペランドとしてアドレスの代わりにこの番号を指定して `call` 命令を実行すると，uBPF ランタイムは対応する関数ポインタを検索し，ヘルパー関数を実行する．

4.3 VM 内情報を取得するための P4 外部関数

P4Shield は共有メモリに格納されている VM 内情報を

```
extern bit<64> get_tcp_mem();

control pipe(inout Headers_t hdr, ...) {
  bit<64> tcp_mem;
  bit<32> threshold;
  bit<32> used_bytes;

  apply {
    if (hdr.ipv4.protocol == IP_PROTO_TCP &&
        (hdr.tcp.flags & TCP_SYN_MASK) != 0 &&
        (hdr.tcp.flags & TCP_ACK_MASK) == 0) {
      tcp_mem = get_tcp_mem();
      threshold = tcp_mem[63:32];
      used_bytes = tcp_mem[31:0];
      if (threshold < used_bytes)
        mark_to_drop();
    }
  }
}
```

図 5: 外部関数を利用する P4 プログラムの例

取得するために，P4 プログラム用の外部関数を提供する．P4 において外部関数は，C 言語で記述されたプログラムなど P4 プログラムの外部で定義されたプログラムを実行するために利用される．P4 プログラムでは外部関数を extern 宣言し，P4 で記述された通常の関数のように呼び出して使用する．必要に応じて引数を取り，値を返却する．各 P4 外部関数は，共有メモリにアクセスするために uBPF ランタイムによって提供されるヘルパー関数である read_vm_info() を呼び出す．このとき，ヘルパー関数に対して必要な VM 内情報に対応する共有メモリのオフセットを指定する．

図 5 は，get_tcp_mem() という外部関数を使用する P4 プログラムの例である．P4 プログラムがパーサブロックでパケットヘッダを解析した後，pipe という名前のコントロールブロックを実行する．コントロールブロックは解析されたパケットヘッダのデータ構造を引数として受け取る．次に，P4 プログラムはパケットが TCP 接続要求用であるかを確認する．具体的には，IPv4 ヘッダでプロトコルが TCP であるか，および TCP ヘッダで制御フラグに SYN が含まれているが ACK が含まれていないことを確認する．これらの条件に当てはまる場合，P4 プログラムは get_tcp_mem() を実行し，ユーザ VM 内の TCP メモリに関する情報を 64 ビット値として取得する．TCP メモリの使用量が閾値を超える場合，P4 プログラムは mark_to_drop() を実行し，パケットを破棄するよう指示する．

P4Shield は，C コンパイラの clang を使用して，C で記述された P4 外部関数を uBPF バイトコードにコンパイルする．生成されたバイトコードは，P4 プログラムをコンパイルして生成された uBPF バイトコードにリンクされる．P4 プログラムをコンパイルするためには，まず P4 リファレンスコンパイラである p4c [18] に含まれる uBPF 向けコンパイラの p4c-ubpf を用いて，P4 プログラムを C 言語のプログラムに変換する．コンパイラを修正することにより，uBPF ランタイムへ新たに追加したヘルパー関数の名前を有する関数ポインタの定義を追加した．この関数ポインタの値は C における関数のアドレスではなく，ヘルパー関数に割り当てられた番号である．次に，変換後の C プログラムを uBPF バイトコードへコンパイルする．外部関数の中でのヘルパー関数の呼び出しは，ヘルパー関数に割り当てられた番号をオペランドとする call 命令にコンパイルされる．

4.4 機密 VM と OVS 間の共有メモリ

P4Shield は 2 種類の共有メモリを使用する．1 つは OVS と P4 VM 間，もう 1 つは P4 VM とユーザ VM 間である．共有メモリオブジェクトを作成するために，P4Shield はホスト上で ivshmem サーバ [20] を実行する．このサーバは VM のデバイスエミュレータである QEMU によって提供される．次に，P4Shield は共有メモリにアクセスするための仮想 PCI デバイスを提供するように VM を構成する．VM のユーザ空間で仮想 PCI デバイスへのアクセスを可能にするために，ivshmem-uio ドライバ [15] を Linux カーネルにロードし，ユーザ空間 I/O (UIO) デバイスを提供する．

OVS は，共有メモリオブジェクトをそのプロセスアドレス空間にマップする．共有メモリにパケットデータを書き込み，共有メモリから P4 プログラムの実行結果を読み取る．P4 VM では，uBPF ランタイムが同一の共有メモリにアクセスするための UIO デバイスをそのプロセスアドレス空間にマップする．このとき，ivshmem-uio ドライバはそのメモリ領域のページテーブルエントリの C ビットをクリアして，共有メモリが暗号化されないようにする．さらに，uBPF ランタイムはユーザ VM との間に確立された共有メモリにアクセスするための別の UIO デバイスをマップし，ユーザ VM が共有メモリに格納した情報にアクセスする．ユーザ VM では，ユーザ空間で動作する情報収集ツールがその共有メモリにアクセスするための UIO デバイスをマップし，P4 プログラムが必要とする情報を共有メモリへ書き込む．また，カーネル内で情報収集ツールを動作させる場合は，共有メモリにアクセスするための仮想 PCI デバイスのメモリ領域をカーネルアドレス空間に再マップする．この再マップされたメモリ領域に対して P4 プログラムが必要とする情報を書き込む．

4.5 P4 プログラム実行状況の確認

P4Shield は仮想スイッチがすべてのパケットを P4 VM に転送して P4 プログラムを実行したかどうかを確認する．そのために，P4 VM 上で動作する各 uBPF ランタイムは，パケットを受信し P4 プログラムを実行する度に共有メモリ上の実行回数カウンタをインクリメントする．そして

表 1: 実験環境

	ホスト 1		ホスト 2	
CPU	Intel Core i7-12700		AMD EPYC 7713P	
メモリ	64 GB		256 GB	
NIC	Intel X540-AT2		Broadcom 57416	
OS	Linux 6.8			
ハイパーバイザ	QEMU-KVM 6.2.0			
	P4 VM	ユーザ VM	P4 VM	ユーザ VM
仮想 CPU	6	4	20	12
メモリ	4 GB	1 GB	32 GB	32 GB
ゲスト OS	Linux 5.15		Linux 6.8	

ユーザ VM のユーザ空間で動作する検知ツールが定期的にこのカウンタを確認し，カーネルが有するパケットの送受信統計データと比較する．単位時間あたりの P4 プログラムの実行回数がユーザ VM のカーネルが処理したパケットの数を下回った場合，届いたパケットに対して P4 プログラムが実行されていない可能性があると判断する．

この判定結果をリングバッファで一定回数保持し，P4 プログラムが実行されていない可能性が一定割合以上検知されたとき，syslog への出力などによりユーザへ通知する．このとき P4 プログラムが実行されていない可能性を検知しても一定回数の判定を待機するのは，ユーザに対する偽陽性のアラート発出を防ぐためである．P4 プログラムの実行回数カウンタとユーザ VM の処理パケット数を厳密に同じ時間で比較するのは難しく，仮想スイッチからユーザ VM に転送する際にパケットドロップが発生する場合もあり，個別の誤検知を防ぐのは難しい．

5. 実験

P4Shield の有効性を示すために，いくつかの実験を行った．まず，P4Shield が VM 内情報を利用する P4 プログラムを用いてパケットフィルタリングを行えることを確認した．また，P4Shield を用いることによって生じるオーバヘッドを調べるために，ユーザ VM の通信性能を測定した．この実験には，**表 1** に示す 2 台のホストを使用し，それらを 10GbE スイッチを介して接続した．各ホストでは，単一の P4 VM とユーザ VM を実行した．

5.1 高度なパケットフィルタリング

仮想スイッチで高度なパケットフィルタリングを実行するため，ユーザ VM で利用可能な TCP メモリが不足することが見込まれる状況において，新規の TCP 接続要求を破棄する P4 プログラムを使用した．この P4 プログラムの一部は 4.3 節で示した．ユーザ VM では Linux カーネルが TCP メモリの使用量を管理し，low，pressure，high の 3 つの閾値を設定している．TCP メモリの使用量が閾値 low 未満の間は，TCP メモリの割当に制限はない．しかし pressure よりも大きくなると，カーネルは TCP メモリの割当を抑制し始める．そして high を超えると，TCP パケットは破棄されるようになる．この実験で使用した P4 プログラムは Linux が有するこのデフォルトポリシーとは異なり，TCP メモリの使用量が独自の閾値を超えた場合に，新たな TCP 接続要求を拒否する．

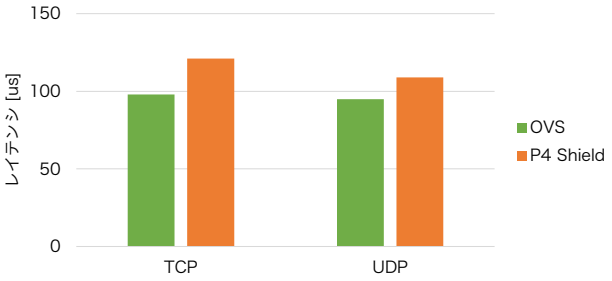

図 6: TCP と UDP のレイテンシ

ユーザ VM では，ユーザ空間で動作する情報収集ツールが P4 VM との間に確立された共有メモリに TCP メモリに関する情報を定期的に格納した．具体的には，/proc/net/sockstat から値を読み取り，独自の閾値と used_bytes の値を共有メモリに記録した．リモートクライアントは，仮想スイッチを介してユーザ VM で実行されているサーバに継続的に SYN パケットを送信し，新規の TCP 接続を要求した．その結果，TCP メモリの使用量が閾値未満である間，クライアントはサーバへの接続に成功した．その一方で TCP メモリの使用量が閾値を超えると，既存の TCP 接続は維持されたまま，新規の TCP 接続要求のみが仮想スイッチによって拒否されることを確認できた．

5.2 通信性能

netperf 2.7.0 [13] を用いて，P4Shield における TCP および UDP のレイテンシとスループットを測定した．リモートホストの netperf クライアントから，ユーザ VM 内の netperf サーバにパケットを送信した．5.1 節で示した P4 プログラムを使用したが，netperf は測定の際に単一の TCP 接続を使用するためわずかな変更を加えた．具体的には，TCP の SYN パケットのみではなく，すべてのパケットに対して VM 内情報を取得するようにした．また，計測パケットが破棄されないよう，ユーザ VM は共有メモリに十分に大きな閾値を格納した．比較として，P4 に対応していない従来の OVS を用いた場合についても測定を行った．この実験では，ホスト 1 を使用した．

クライアントに 1 バイトのメッセージサイズを指定したときの，90 パーセンタイル往復レイテンシを**図 6** に示す．従来の OVS と比べると，P4Shield のレイテンシは TCP で 23%，UDP で 15%増加した．これは P4 VM へのパケットの受け渡し，P4 プログラムの実行，VM 内情報取得にかかるオーバヘッドである．

メッセージサイズを約 64KB まで増やしたときのスルー

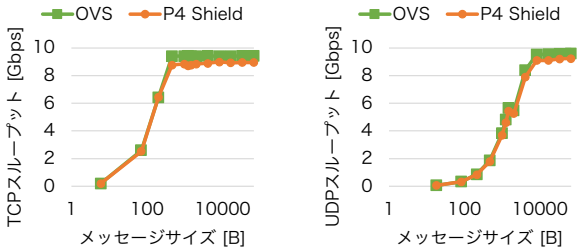

図 7: TCP と UDP のスループット

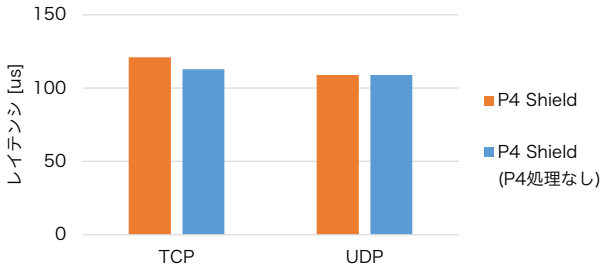

図 8: P4 処理によるレイテンシへの影響

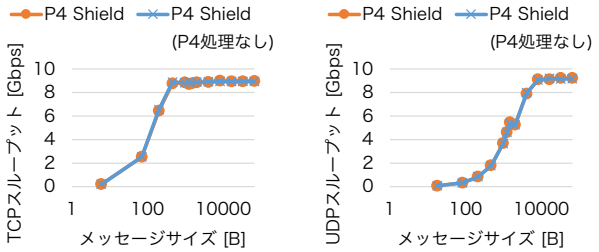

図 9: P4 処理によるスループットへの影響

プットを**図 7** に示す．P4 Shield におけるスループットの低下は TCP で平均 5%，UDP で平均 4%に留まった．スループットのこの穏やかな低下は仮想スイッチでの並列処理に起因していると考えており，これによって P4 スイッチのオーバヘッドが隠蔽されている，

次に，通信性能に対する P4 プログラム実行の影響を調査した．比較のために，P4 VM に P4 プログラムをロードしない場合についても測定を行った．この場合，P4 VM から仮想スイッチに対しては P4 プログラムの実行結果の代わりに常にパケットを転送させるように結果を返すようにした．P4 プログラムの実行がレイテンシに及ぼす影響を**図 8** に示す．TCP ではレイテンシを 7%増加させた一方で，UDP では有意な差は見られなかった．同様に，スループットへの影響を**図 9** に示す．TCP，UDP ともにほとんど影響は見られなかった．

最後に，P4 プログラムの実行時間の内訳を調査した．各処理時間の中央値を**図 10** に示す．総実行時間の 68%を外部関数の実行以外の P4 処理が占めており，その他の 27%は外部関数内で実行される VM 内情報の復号にかかる時間であることが分かった．しかし，総実行時間はわずか 421

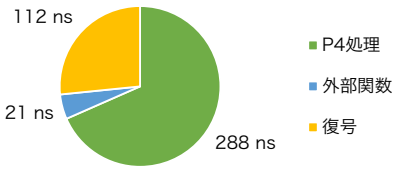

図 10: P4 プログラム実行時間の内訳

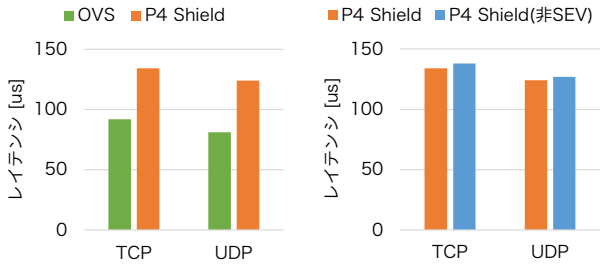

図 11: 機密 VM を使用した P4 Shield におけるレイテンシ

ns に留まっており，レイテンシの増分よりもはるかに小さいものであった．これは，P4 Shield のオーバヘッドのほとんどが OVS と P4 VM 間の通信に起因しているということである．

5.3 機密 VM 使用時の通信性能

P4 VM およびユーザ VM を AMD SEV を使用して機密 VM として実行したときの，TCP および UDP 通信のレイテンシとスループットを測定した．この実験では，ホスト 2 を使用した．**図 11**(左) に示すように，レイテンシは TCP で 46%，UDP で 53%増加した．これらの増加は，ホスト 1 の結果よりも遥かに大きい．これが機密 VM のオーバヘッドによって生じた差であるかを調べるために，P4 VM を通常の VM として実行したときのレイテンシを調査した．その結果，図 11(右) に示すようにレイテンシは減少せず，P4 VM を機密 VM として動作させるオーバヘッドは無視できる程度であることがわかった．同様にユーザ VM も非機密 VM として実行したが，レイテンシの差はごくわずかであった．これはすなわち，本実験で見られた P4 Shield のオーバヘッドはホスト 1 とホスト 2 における他の違いに起因しているということである．

同様にして測定したスループットを**図 12** に示す．クライアントがメッセージサイズを増加させても従来の OVS でさえ 10 Gbps に近づくことはなく，十分な性能を達成できていない．TCP では，従来の OVS のスループットは平均では P4 Shield よりも 4%高い一方で，ほとんどのメッセージサイズにおいて P4 Shield のスループットのほうが高くなっている．UDP では，ほぼ常に P4 Shield のスループットのほうが高い結果となっている．また，メッセージサイズが大きくなると，スループットは急激に減少している．これらの傾向は，P4 VM とユーザ VM を通常の VM

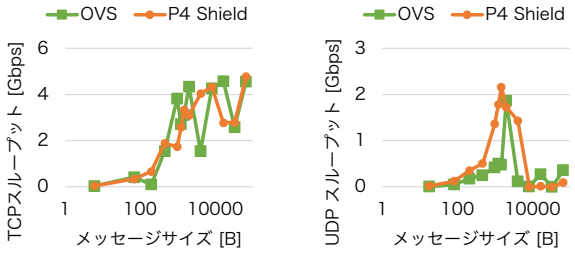

図 12: 機密 VM を使用した P4 Shield における スループット

として実行したときも同様であった．これらの結果から，ホスト 2 は何らかの理由により OVS を使用して十分な性能を達成できなかったと結論づける．

5.4 P4 プログラムの実行確認

クラウドがユーザの P4 プログラムを実行しないという攻撃をユーザ VM において検知できることを確認する実験を行った．リモートホストからユーザ VM 上の Web サーバに対して HTTP リクエストを送出し続けている状況で，仮想スイッチにおいて一時的に P4 プログラムの実行をバイパスさせた．この実験では，ホスト 2 および 5.2 節で示した P4 プログラムを使用した．その結果，仮想スイッチが P4 プログラムの実行をバイパスしている間，ユーザ VM のユーザ空間で動作する検知ツールがその攻撃を検知し，ユーザに対してアラートを発出できることを確認した．

5.5 安全性評価

クラウドは，ユーザの P4 プログラムや P4 プログラムによって取得された VM 内情報を盗聴および改ざんする攻撃を試みる可能性がある．P4 Shield では P4 プログラムは P4 VM に隔離されており，そのメモリとレジスタは TEE によって保護されている．これにより，クラウドがユーザの P4 プログラムとユーザ VM の機密性や完全性を侵害することを防ぐ．ただし，TEE はユーザ VM から P4 VM に情報を渡すために使用する共有メモリを保護できない．したがってこの通信の機密性を保証するために，P4 Shield ではユーザ VM 内で情報を暗号化し，P4 VM 内で復号する．完全性は MAC を使用することで検証する．

仮想スイッチ内で実行される P4 ランタイムに脆弱性が存在する場合，ユーザは悪意ある P4 プログラムをロードすることで仮想 P4 スイッチに対する攻撃を試みる可能性がある．また，パケット転送の遅延を引き起こすために，仮想スイッチ内で大量のリソースを消費する P4 プログラムを実行する可能性もある．P4 Shield ではユーザの P4 プログラムは P4 VM 内に閉じ込められ，仮想スイッチから分離されている．したがって，P4 プログラムが異常な動作をしたとしても，仮想スイッチに影響を及ぼすことはできない．さらに P4 Shield では，P4 プログラムを実行する uBPF ランタイムのロード時検証機能により，不正なアクセスと無限ループの可能性を排除している．

クラウドは，ユーザの P4 プログラムを実行しないことにより，容易に無効化することが可能である．その結果，破棄されるべきパケットをユーザ VM に転送したり，転送されるべきパケットを破棄したりする可能性がある．P4 Shield では，P4 VM が P4 プログラムの実行ログを記録することでこの攻撃を検出する．具体的には，ユーザ VM がこのログをパケット送受信の統計と比較することで，P4 プログラムが適切に実行されていない可能性を検出する．P4 Shield では，P4 VM からユーザ VM へログを渡すために使用する共有メモリも，暗号化と MAC によって保護する．ただし，現段階では暗号化と MAC による保護は未実装である．

6. 関連研究

P4rt-OVS [17] は OVS ベースの仮想 P4 スイッチである．P4 プログラムは ovs-ofctl コマンドを用いて仮想スイッチにロードされ，パケット到着時に仮想スイッチ内で実行される．また，p4c-ubpf を用いて uBPF バイトコードにコンパイルされる．P4 Shield とは異なり，仮想スイッチ内で uBPF ランタイムが動作する．uBPF により仮想スイッチは P4 プログラムからある程度保護される．しかし，uBPF ランタイムに脆弱性が存在する場合，仮想スイッチ自体が侵害される．また，P4rt-OVS は管理者が P4 プログラムを仮想スイッチにロードすることを想定している．ユーザが P4 プログラムをロードしたり，ユーザの VM 内部の情報を使用したりすることはできない．

SGX-Box [10] は TEE の 1 つである Intel SGX を用いて，ミドルボックス内で暗号化トラフィックを安全に復号することを可能にしている．AMD SEV とは異なり，SGX はアプリケーションがエンクレイヴと呼ばれる保護されたメモリ領域を作成する．SGX-Box は，SGX エンクレイヴ内に保持するセッション鍵を用いてパケットを復号し，侵入検知などを行う．また，暗号化トラフィックを処理するための SB lang と呼ばれる高レベルプログラミング言語を提供している．SB lang は TLS ハンドシェイクの解析，パケット再構成，復号・再暗号化などの詳細を開発者から隠蔽する点で P4 言語と類似している．また，ネットワーク管理者が SB lang で記述されたプログラムを SGX エンクレイヴにロードすることを想定している．

EndBox [9] は Intel SGX を用いて，ファイアウォールや侵入検知システムなどのミドルボックスの機能をクライアント側で安全に実行することを可能にしている．クライアント側で動作する EndBox クライアントは SGX エンクレイヴ内に暗号通信のための鍵を保持し，組織やプロバイダ内の EndBox サーバに VPN を用いて接続する．クライ

アントのアプリケーションは EndBox サーバを経由しない限り通信を行うことができないため，EndBox の利用を強制することができる．

VM 内の情報を用いたきめ細かいパケットフィルタリングが提案されている．VMwall [19] は Xen の管理 VM 内で実行され，VM イントロスペクションを用いて VM のメモリを解析する．パケットヘッダのポート番号を基にそのパケットを送信したプロセスまたは受信するプロセスを見つけ，そのプロセス名がホワイトリストになければパケットを破棄する．xFilter [14] は VMwall に似ているが踏み台攻撃を対象としている．VM イントロスペクションを使用してパケットを送信したプロセスやユーザの ID を取得し，動的にフィルタリングルールを生成して追加することができる．これらのシステムで使用される VM 内情報は，P4 Shield の P4 プログラムでも使用できる．

7. まとめ

本稿では，VM 内情報を利用できるように拡張した P4 プログラムをユーザごとに用意される P4 VM で安全に実行する P4 Shield を提案した．P4 Shield では，仮想スイッチが受信したパケットを対応する P4 VM に転送して P4 プログラムを実行し，実行結果に基づいてパケットの転送処理を行う．P4 Shield は P4 VM を機密 VM として実行することでクラウドからユーザの P4 プログラムを保護し，P4 VM 内に作成された uBPF サンドボックスにユーザの P4 プログラムを隔離することでクラウドの仮想スイッチを保護する．共有メモリに格納されたユーザ VM 内の情報は，P4 プログラムの外部関数を介して uBPF ランタイムのヘルパー関数を実行して取得する．P4 Shield を Open vSwitch と uBPF ランタイムに実装，動作確認および性能測定を行った．

今後の課題の 1 つは，機密 VM をサポートする AMD EPYC プロセッサを搭載したホストでの通信性能の改善である．Intel Core プロセッサを搭載したホストでは遥かに高い性能を示したため，前者のホストでのボトルネックを調査する必要がある．また，P4 プログラムでどの情報を利用できるようにすべきか，どのような API を提供すべきかを検討する．

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] BPF Documentation — The Linux Kernel documentation. https://www.kernel.org/doc/html/latest/bpf/.
[2] GitHub - iovisor/ubpf: Userspace eBPF VM. https://github.com/iovisor/ubpf.
[3] Open vSwitch. https://www.openvswitch.org/.
[4] P4 – Language Consortium. https://p4.org/.
[5] uBPF: Main Page. https://iovisor.github.io/ubpf/.
[6] Advanced Micro Devices, Inc. Secure Encrypted Virtualization API Version 0.24, 2020.
[7] Mihai Valentin Dumitru, Dragos Dumitrescu, and Costin Raiciu. Can we exploit buggy p4 programs? In *Proceedings of the Symposium on SDN Research*, SOSR '20, pp. 62–68, New York, NY, USA, 2020. Association for Computing Machinery.
[8] Tal Garfinkel, Mendel Rosenblum, et al. A virtual machine introspection based architecture for intrusion detection. In *Ndss*, Vol. 3, pp. 191–206. San Diega, CA, 2003.
[9] David Goltzsche, Signe Rüsch, Manuel Nieke, Sébastien Vaucher, Nico Weichbrodt, Valerio Schiavoni, Pierre-Louis Aublin, Paolo Cosa, Christof Fetzer, Pascal Felber, Peter Pietzuch, and Rüdiger Kapitza. EndBox: Scalable Middlebox Functions Using Client-Side Trusted Execution. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 386–397, June 2018.
[10] Juhyeng Han, Seongmin Kim, Jaehyeong Ha, and Dongsu Han. SGX-Box: Enabling Visibility on Encrypted Traffic using a Secure Middlebox Module. In *Proceedings of the First Asia-Pacific Workshop on Networking*, APNet '17, pp. 99–105, New York, NY, USA, 2017. Association for Computing Machinery.
[11] Intel Corp. DPDK – The Open Source Data Plane Development Kit. https://www.dpdk.org/.
[12] Intel Corp. Intel Trust Domain Extensions, 2023.
[13] R. Jones. Netperf Benchmark. https://hewlettpackard.github.io/netperf.
[14] Kenichi Kourai, Takeshi Azumi, and Shigeru Chiba. Efficient and fine-grained vmm-level packet filtering for self-protection. *International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS)*, Vol. 5, No. 2, pp. 83–100, 2014.
[15] C. Macdonell. ivshmem-uio. https://github.com/shawnanastasio/ivshmem-uio.
[16] Open Programmable Infrastructure Project. IPDK Networking Recipe (P4 Control Plane). https://github.com/ipdk-io/networking-recipe.
[17] Tomasz Osiński, Halina Tarasiuk, Paul Chaignon, and Mateusz Kossakowski. P4rt-OVS: Programming Protocol-Independent, Runtime Extensions for Open vSwitch with P4. In *2020 IFIP Networking Conference (Networking)*, pp. 413–421, June 2020.
[18] P4 Compiler Project. P4C. https://github.com/p4lang/p4c.
[19] Abhinav Srivastava and Jonathon Giffin. Tamper-Resistant, Application-Aware Blocking of Malicious Network Connections. In Richard Lippmann, Engin Kirda, and Ari Trachtenberg, editors, *Recent Advances in Intrusion Detection*, pp. 39–58, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
[20] The QEMU Project Developers. Inter-VM Shared Memory Device. https://www.qemu.org/docs/master/system/devices/ivshmem.html.