Vol.2021-OS-151 No.7
2021/3/2

Lua言語処理系を組み込んだハイパーバイザによる柔軟で高セキュアなSDN基盤

尾内 智哉[1,a)]　並木 美太郎[1,b)]

概要：既存のネットワークの課題を解決するためにSDNが使用されている．しかし, SDNはSDNコントローラが単一障害点になるという問題がある．DoS攻撃の緩和やSDNコントローラへのアクセス制御によってSDNコントローラのセキュリティを強化する研究があるがOSが乗っ取られた場合は効果がなくなる．本研究ではBitVisorの保護ドメインで動作するSDNコントローラを設計して, ネットワークの監視と制御を行う高セキュアなSDN基盤を提案する．また, 保護ドメインを利用したC言語でなく先行研究のLVisorで動作するスクリプト言語処理系のLuaで記述した保護ドメインで動作するSDNコントローラを提案する．ネットワークを監視してゲストOSが接続されたSDNスイッチから得た情報をもとにSDNにより柔軟にネットワークを制御する, 迅速なインシデント対応を行うセキュアな仮想ネットワークを構築する．LVisorではゲストOSのネットワークのみを監視したがSDNコントローラを導入することで, より広範なネットワークを監視・制御できるようになる．今後の課題はLuaで記述したSDNコントローラを保護ドメインで動かすことである．

1. はじめに

既存のネットワークの課題として, 物理ネットワークの構成の変更が大変であること, ネットワーク機器のコストが高いこと, そしてネットワーク機器のベンダーロックインが挙げられる．そこでSDNによってネットワークをソフトウェアで制御することで, これらの課題を解決した．OpenFlow[1]はSDNを実現するネットワークプロトコルの1つである．ネットワークの制御と転送を, それぞれコントロールプレーン（SDNコントローラ）とデータプレーン（SDNスイッチ）で行うことによってネットワークをソフトウェアで集中制御する．しかし, SDNコントローラが単一障害点になるという新たな問題が生じた．対策として, DoS攻撃の緩和[2]やNorthbound APIのアクセス制御[3]によってSDNコントローラのセキュリティを強化する研究が行われている．しかし, 高い特権レベルで動作するOSやハイパーバイザが乗っ取られたら場合, これらの対策は効果がなくなる．

OSは計算機が動作するのに必要不可欠なソフトウェアである．CPU, メモリ, ディスク, NICなどのハードウェアの管理し複雑な要素を抽象化してユーザに提供する．また, 複数のアプリケーションを動かすためにプロセスの管理やスケジューリングなどを行う．OSは管理するハードウェアからの割り込みやアプリケーションから渡されるデータの処理など複数のイベントを同時に処理する．その複雑さから, OSでないソフトウェアと比べてOSからは多くのバグが発見されている[4]．OSのバグは重大な脆弱性に繋がる可能性もあり, OSの欠陥による重要なデータの流出や消失に繋がりうるため信頼できる必要がある．医療機器や車載機器などのミッションクリティカルなシステムではシステムの欠陥が人の命に関わる事故につながるため, 搭載するOSはセキュアでバグが少ないことが求められる．しかし, OSのコードは増加し続けており, 現在もOSからは多くのバグが発見されている．

ハードウェア技術の進歩によってSSDや不揮発性メモリなどの次世代ハードウェアが登場している．OSは新しいハードウェアが登場するたびに, これらのハードウェアを制御するためのコードが追加される．モノリシックカーネルであるLinuxはコードが増加し続けている．一番高い特権レベルのRing 0で動作するプログラムも増加し, 攻撃層も大きくなる．OSの危殆化は, OSの上で動作するすべてのアプリケーションに影響を及ぼすためセキュリティの保証が難しくなる．

本研究では汎用的なOSが肥大化し攻撃層が大きいことと, 修正されていないバグが残されているということから脆弱であると考え, SDNコントローラを軽量なハイパーバイザBitVisorの保護ドメインで動作させる高セキュアな

[1] 東京農工大学
Tokyo University of Agriculture and Technology
[a)] s178565q@st.go.tuat.ac.jp
[b)] namiki@cc.tuat.ac.jp

SDN 基盤を提案する．また市川による LVisor[12] を拡張した, C 言語でなくスクリプト言語処理系の Lua[13] で記述し, VMX root モードの Ring 3 で動作する SDN コントローラを方法を提案する．汎用的な OS と比較して攻撃層が小さく, 実装がシンプルなハイパーバイザで SDN によるネットワークの制御を行うことで OS に依存せずに, OS が信頼できない場合でもセキュアに動作させることができる．Lua で SDN コントローラを動作させることができればハイパーバイザの実行時に動的にディスクまたはネットワーク経由でプログラムを取得して実行することができる．ハイパーバイザとゲスト OS が動作する特権レベルの違いから安全に SDN コントローラの実行が可能となる．このように, 従来のように OS の上でアプリケーションを動作させるのではなくハイパーバイザはセキュリティの観点からアプリケーションの有用な実行基盤になる．

2. 関連研究

Chen は OS の上で動作するアプリケーションの一部をハイパーバイザに再配置し, ハイパーバイザをアプリケーションの実行基盤として提供するアーキテクチャを提案した [5]．OS は巨大で攻撃を受けやすいことから, 小さくシンプルであるハイパーバイザにロギング, 侵入防止, 侵入検知などのアプリケーションを再配置することを主張した．OS はハイパーバイザでのアプリケーションの実行は OS に依存せずに動作し, 信頼できない OS であってもハイパーバイザがより高い特権レベルで動作するためセキュアである．また, OS の変更を必要としないためアプリケーションを容易に追加することができる．

OS のセキュリティの強化のためにハイパーバイザを用いた研究が行われている．品川は準パススルー型アーキテクチャの BitVisor を提案した [6]．BitVisor は VMX root モードの Ring 3 でアプリケーションを動作させる保護ドメインという機能がある．メモリ空間が分離されておりセキュアにプログラムを動作させることができる．保護ドメインでネットワークとディスク I/O をゲスト OS から透過的に暗号化することで OS のセキュリティを強化する．

ネットワークの監視では BitVisor でパケットフィルタリングを行い, パケットの送信元での DDoS 攻撃の防止に応用されている [7]．BitVisor に eBPF の VM を移植して柔軟なパケットフィルタリングを行い, パケットの送信源での DDos 攻撃を防止した．

市川はハイパーバイザをマルウェア解析の基盤として BitVisor にスクリプト言語処理系の Lua と LibVMI を移植するアーキテクチャの LVisor を提案し, BitVisor の保護ドメインで Lua を動作させることに成功した [10], [11], [12]．これによって, プログラマブルなハイパーバイザを実現しマルウェア解析における柔軟パターンマッチングができるようになった．ゲスト OS から取得できる情報とハイパーバイザから取得できるゲスト OS の情報にはセマンティックギャップがあるという問題を LibVMI によって解決し, スクリプト言語処理系の Lua を使用することでスクリプト言語で記述したプログラムによる柔軟なパターンマッチングを行えることを示した．

3. 設計

提案するシステムの構成図を図 1 に示す．

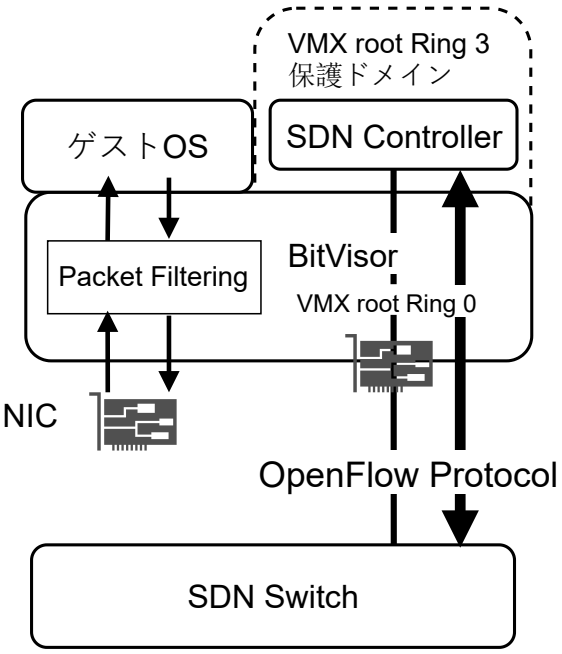

図 1 提案システム

ハイパーバイザは軽量なハイパーバイザの BitVisor を使用する．BitVisor でパケットフィルタリングによってネットワーク監視を行い, 保護ドメインで SDN コントローラを動かすことでネットワークを制御する．

図 1 の SDN コントローラは従来の OS の上で動作する SDN コントローラとは異なり, OS の下でハイパーバイザと同じ VMX root モードで動作する．ゲスト OS とハイパーバイザが動作する特権レベルの違いから攻撃層の大きいゲスト OS が危殆化しても, その影響はハイパーバイザまでには及ばないため SDN コントローラはセキュアとなる．ネットワークの監視によって得た情報をもとに SDN によりネットワークを制御するという連携も可能となる．

また, 市川による LVisor[12] を拡張したスクリプト言語処理系の Lua で記述し, VMX root モードの Ring 3 で SDN コントローラを動作させる．スクリプト言語処理系を使用することで状況に応じて実行する SDN コントローラのプログラムを状況に応じて切り替えることが可能となる．LVisor に移植された LibVMI によってゲスト OS の情報を取得することができれば, 保護ドメインで動作する SDN コントローラによりネットワークを制御してマルウェアに感

染したゲスト OS から透過的にネットワークを制御することができる.

3.1 ハイパーバイザでのネットワークの監視

ハイパーバイザでゲスト OS のネットワークを監視するために 1 パケットずつパケットをフィルタリングする. パケットフィルタリングのルールを設定する方法を次の図 2 に示す. パケットをフィルタリングするルールはハイパーバイザを管理する外部の計算機からネットワーク経由で設定するか, ゲスト OS からハイパバイザーコールによってフィルタリングルールを設定する. また, ルールは LVisor の Lua でも設定可能である. フィルタリングのルールは IP アドレスと TCP ポート番号をとして, ゲスト OS を経由するパケットが設定した IP アドレスと TCP ポート番号に一致したときにパケットをドロップし通信を遮断する.

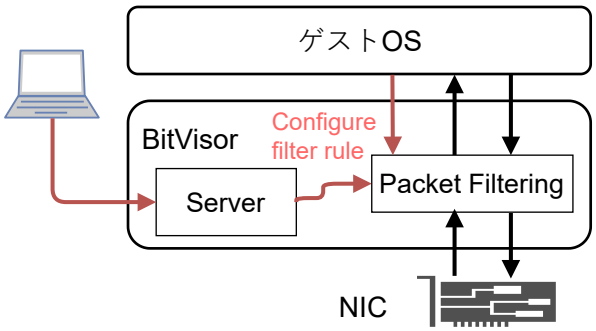

図 2 パケットのフィルタリング

3.2 保護ドメインで SDN コントローラの実現

BitVior は VMM のコアと切り離して, プログラムを実行する保護ドメインがある. 例として idman による IC カードを用いたユーザ認証やゲスト OS から透過的なディスク, ネットワーク I/O の暗号化は保護ドメインで実行される. 保護ドメインは VMX root モードの Ring 3 で動作するプログラムであり, ハイパーバイザとしての役割を果たす BitVisor のコアと切り離している. 一番高い特権レベルで動作する最小限にすることで保護ドメインで実行されるプログラムにバグがあり, クラッシュしたとしてもその影響が VMM のコアまで及ばなくなる. BitVisor はこのようにしてハイパーバイザとしての機能を提供しつつ, 暗号化やユーザ認証といったハイパーバイザとは関係のない処理は保護ドメインで動作させることで BitVisor の安全を保証する.

保護ドメインはメッセージインタフェースという仕組みがある. これは保護ドメインに割り当てあられたディスクリプタを通じてデータをやり取りするものである. BitVisor は保護ドメインが使用するためのシステムコールが実装されており, これらのシステムコールを使って通信用のディスクリプタの生成やデータの送信, 保護ドメインの作成を行う. 保護ドメインはゲスト OS より高い特権レベルで動作することとゲスト OS から干渉されないことから機密環境としての一面もあり, 耐タンパデバイスの高速化に応用されている [9].

ネットワーク制御を行う SDN コントローラは保護ドメインで動作させることで, ゲスト OS より高い特権レベルで OS に依存せずに動作することから OS の危殆化による影響を受けずにセキュリティを向上できる.

SDN コントローラはネットワークを扱うため, BitVisor に組み込まれた TCP/IP プロトコルスタックの lwIP を使用して OpenFlow を扱うプログラムを実装する. 保護ドメインで SDN コントローラを動作させるために, lwIP の Socket API を保護ドメインにシステムコールとして提供する.

また, LVisor の Lua で記述して VMX root モードの Ring 3 で SDN コントローラを動作させる. これによって SDN コントローラのプログラムを BitVisor の実行中にディスクから読み込み, 動作させる.

提案するシステムでは SDN スイッチには複数の計算機が接続されており, BitVisor 上またはベアメタルで動作する. SDN スイッチに接続された複数の計算機が SDN による仮想ネットワーク上で通信する. これらの計算機は BitVisor 上では動作してもしなくても良いが BitVisor を使用することでセキュリティの向上が期待できる. このネットワークでは BitVisor の保護ドメインで動作する SDN コントローラが 1 つあり SDN スイッチを制御することで SDN スイッチに接続された計算機への攻撃から守る. 攻撃を検出するとゲスト OS への通知を行い, SDN コントローラが SDN スイッチを制御して攻撃を行う計算機からの通信を遮断する.

3.2.1 保護ドメインで利用可能な Socket API

BitVisor の lwIP はデフォルトで raw API によるコールバック形式の関数を使用してネットワークアプリケーションを動かす. raw API を使用する lwIP はコールバック形式のため Ring 0 と密接に関係していて保護ドメインでネットワークを使用するプログラムの動作が難しかった. しかし, Socket API を使う lwIP は汎用的な OS が提供するようなソケットを使ったプログラムが書けるため, これらの関数をシステムコールとして保護ドメインに提供することで保護ドメインで動作する SDN コントローラが実現できる.

Socket API を使う lwIP の動作のためにはマルチスレッドで動作するため, プラットフォームごとにセマフォ, ミューテックス, スレッド間でデータのやり取りを行うための mail box というキューのようなデータ構造を実装する必要がある. 次の表 1 に Socket API を使用するためにプラットフォームごとに定義する API を示す.

表 1 Socket API の lwIP で使用するために必要な API

API	機能
sys_init	sys_arch レイヤの初期化
sys_sem_new	新しいセマフォの作成
sys_sem_free	セマフォの開放
sys_sem_signal	セマフォをインクリメント
sys_arch_sem_wait	セマフォがインクリメント されるまでスレッドをブロック
sys_sem_valid	セマフォが有効化されているか確認
sys_sem_set_invalid	セマフォを無効化
sys_mutex_new	新しいミューテックスを作成
sys_mutex_free	ミューテックスを開放
sys_mutex_lock	ミューテックスをロック
sys_mutex_unlock	ミューテックスをアンロック
sys_mutex_valid	ミューテックスが有効化されているか確認
sys_mutex_set_invalid	ミューテックスを無効化
sys_mbox_new	新しい空の mail box を作成
sys_mbox_free	mail box を開放
sys_mbox_post	mail box にメッセージを送信
sys_arch_mbox_fetch	mail box からメッセージを取得.
sys_arch_mbox_tryfetch	mail box からメッセージを取得. ない場合は即座に終了.
sys_mbox_valid	mail box が有効化されているか確認
sys_mbox_set_invalid	mail box を無効化
sys_thread_new	新しいスレッドを作成
sys_arch_protect	危険領域の保護
sys_arch_unprotect	危険領域の保護を解除
sys_now	ミリ秒単位での現在時刻の取得

raw API を使用する lwIP はシングルスレッドで動作するため OS がない環境でも動作するが Socket API を使用する lwIP はマルチスレッドで動作するため OS によるスレッドのスケジューリング必要とする．BitVisor ではハイパーバイザ自体は単一もメモリ空間で動作し, VMX root モードの Ring 0 で動作するプログラムはスレッドである．スレッドは non-preemptive なスケジューリングを採用しており, 各スレッドは自分で schedule() を呼び出しスケジューリングを行う．スレッドは thread_new() で作成することができ BitVisor ではタイマー用のスレッドや telnet サーバが動作する．このように BitVisor にはスレッドを作成しスケジューリングを行う機能が実装されているため BitVisor のスレッドを使い Socket API を使う lwIP を動作させることができる．

表 1 に示す API を実装し Socket API を有効化することで汎用的な OS にある socket(), bind(), listen() などを使用したソケットプログラミングが行えるようになる．Socket API の lwIP で使用する API を次の表 2 に示す．

これをシステムコールとして保護ドメインに提供することで保護ドメインでネットワークを扱うアプリケーションを動かすことができる．これと LVisor で VMX root モードの Ring 3 で動作する Lua を組み合わせることでスクリプト言語で書いた SDN コントローラを実現できる．

表 2 Socket API の lwIP で使用する API

API	機能
lwip_accept	コネクションを待ち受ける
lwip_bind	IP アドレスとポート番号を ローカルホストに紐付ける
lwip_shutdown	全二重通信の一部を閉じる
lwip_getpeername	リモートのクライアント オブジェクトの情報を返す
lwip_getsockname	オブジェクトに紐付いたアドレス情報を返す
lwip_setsockopt	ソケットのオプションの設定
lwip_connect	遠隔のホストとコネクションを確立する
lwip_listen	コネクションを待ち受ける
lwip_recv	データを受信
lwip_send	データを送信
lwip_sendmsg	データを送信
lwip_socket	ソケットの作成
lwip_select	I/O の多重化
lwip_ioctl	ブロッキング/ノンブロッキングを設定
lwip_read	データを受信
lwip_write	データを送信
lwip_close	コネクションを閉じる
lwip_fnctl	ソケットの設定の取得や変更

3.2.2 Lua で記述した SDN コントローラ

スクリプト言語処理系で記述して保護ドメインで動作する SDN コントローラを実現するために, VMX root モードの Ring 3 で動作する LVisor の Lua を使用して SDN コントローラを実現する．これによってネットワークを制御するルールを Lua で書くことができる．システムコールとして提供した Socket API の lwIP を Lua から使用することで SDN コントローラを実装する．SDN コントローラのプログラムはディスクから読み込み, 状況に応じて実行するプログラムを切り替える．

保護ドメインのプログラムは BitVisor のバイナリに ELF バイナリを埋め込み動作させる．このためプログラムの修正には BitVisor をコンパイルし直す必要があり, 非効率な面がある．これをスクリプト言語処理系の Lua を使用することで解決する．スクリプト言語のため, BitVisor に SDN コントローラのプログラムのバイナリを埋め込む必要がなくなり, プログラムの変更のたびに BitVisor のバイナリを生成し直す必要がなくなる．また, Lua はガベージコレクションを備えているため C 言語のようにメモリ管理をプログラマが行う必要がなくシンプルにプログラムを記述できることから生産性, 可読性が向上する．

3.2.3 Lua の socket の実装

Lua は他のスクリプト言語処理系の Lisp, Ruby, Python と比較してバイナリのサイズが小さく依存ライブラリも libc と libm だけで少ないため, ゲーム機器やネットワーク機器に組み込まれて使用されている．スクリプト言語とし

て十分に成熟しており小さなフットプリントと容易にアプリケーションに組み込めることから広く利用されている. しかし, 必要最低限な機能しか備えていないため標準では socket を実装していない. このため Lua から socket を使用するために拡張ライブラリの luasocket を LVisor の Lua に組み込む.

4. 実装

表 1 に示すセマフォや mail box などを扱う API を実装することで Socket API を使用する lwIP を動作させた. そして, 表 2 をシステムコールとして保護ドメインに提供することで保護ドメインでのソケットを使用できるようにした. このソケットを使い C 言語で記述して保護ドメインで動作する SDN コントローラを実装した. 最終的には, 保護ドメインで動作する LVisor の Lua から Socket API を使用する lwIP の使用して Lua で SDN コントローラを動作させることが目標である.

4.1 SDN コントローラの実装

Socket API を使用する lwIP をシステムコールとして保護ドメインに提供して, ソケットを使い OpenFlow プロトコルを扱う保護ドメインで動作する SDN コントローラを実装した.

性能の追求や完璧な OpenFlow の実装が目的ではないため, 初期のバージョンの 1.0 を採用し, OpenFlow を扱うための必要最低限の OpenFlow メッセージを実装する. 次の表 3 に実装した 4 つの OpenFlow メッセージを示す.

表 3 OpenFlow メッセージ

OpenFlow メッセージ	目的
OFPT_HELLO	使用する OpenFlow のバージョンを決定
OFPT_ECHO_REPLY	死活監視, 接続確認のために送信される Echo Request への返答
OFPT_FLOW_MOD	OpenFlow スイッチへのフローエントリの追加、変更
OFPT_PACKET_OUT	Pakcet In に対するパケットの出力

4.2 ハイパーバイザでのパケットフィルタリング

BitVisor でのパケットフィルタリングは NIC が扱うシャドウバッファとゲスト OS が扱うバッファ間でパケットデータをコピーする前に 1 パケットずつフィルタリングを行いパケットを通すかドロップするか決定する. フィルタリングを行うプログラムを次の program 1 に示す.

フィルタリングを行うルールは IP アドレスと TCP のポート番号であり, ルールの設定は管理用のマシンからネットワーク経由, またはゲスト OS からのハイパーバイザコールによって行うことができる.

program 1 パケットフィルタリングを行うプログラム

```
for (int i = 0; i < host_count; i++) {
  if ((iphdr->src.addr == ipaddrs[i].addr \
    && htons(tcphdr->src) == ports[i]) ||
    (iphdr->src.addr == ipaddrs[i].addr \
    && htons(tcphdr->dest) == ports[i]) ||
    (iphdr->dest.addr == ipaddrs[i].addr \
    && htons(tcphdr->src) == ports[i]) ||
    (iphdr->dest.addr == ipaddrs[i].addr \
    && htons(tcphdr->dest) == ports[i])) {
    return 0;
  }
}
```

4.3 Preemption timer による強制的な VM Exit

BitVisor ではハイパーバイザであるため, ゲスト OS でセンシティブな命令が実行され VM Exit が発生したときに動作する. このため VM Exit が発生して BitVisor に制御が移り, スケジューラによって lwip のスレッドに CPU が割り当てられないと SDN コントローラは動作しない.

実験でゲスト OS として使用した OS の Ubuntu 18.04 ではバックグラウンドで多くのプロセスが動作するためユーザが何もしていない状態でも VM Exit は発生し, SDN コントローラは正常に動作した. しかし, ゲスト OS の利用者が悪意を持ってゲスト OS のバックグラウンドプロセスを止めて VM Exit が発生しないようにした場合 SDN コントローラは動作できなくなる. この問題は Intel VT-x の preemption timer という機能を利用することで解決できる. SDN コントローラはフローエントリに一致しないパケットが SDN スイッチに入り, Packet In によって SDN コントローラに処理方法を問い合わせたときのみ動作すればよく, 高い性能を追求するわけではないため今回はこの機能は使用していない.

5. 評価

提案システムの BitVisor のネットワーク監視によるオーバーヘッドの計測とポートスキャンの検出と遮断による保護ドメインで動作する SDN コントローラの動作確認を行う.

5.1 評価環境

実験環境を次の表 4 に示す.

5.2 ゲスト OS のネットワークの監視のオーバーヘッド

ハイパーバイザでのネットワーク監視のためのパケットフィルタリングでは 1 パケットずつ, パケットを通すかドロップするかを決定する. BitVisor とパケットフィルタリングを導入することによってベアメタル時のパフォーマンスと比較して, どの程度性能が劣化するか, ゲスト OS のネットワークのオーバーヘッドを iperf3 で計測した. ネッ

表 4 実験環境

Hypervisor	BitVisor 2.0
OS	Ubuntu 18.04, Linux Kernel 5.05
CPU	Intel (R) Core (TM) i7- CPU 1.80GHz
RAM	8GByte
NIC	Intel Corporation Ethernet Connection 219-V (rev 21)
	Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller (rev 06)
SDN switch	Open vSwitch 2.13.1

トワークのオーバーヘッドを計測した結果を次の 3 に示す.

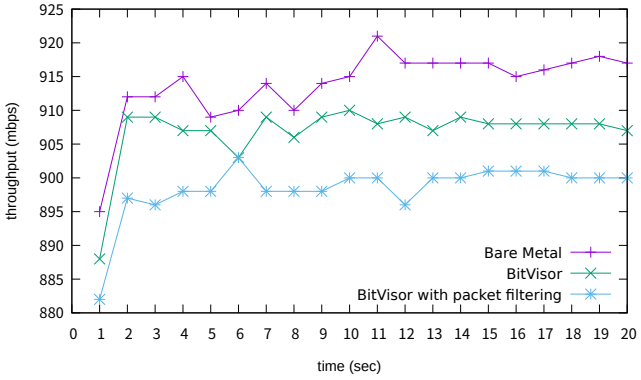

図 3 iperf3 で計測したネットワークスループット

iperf3 で OS のネットワークのスループットを計測した結果, BitVisor を使用しない場合は 913.9Mbps, 変更を加えていない BitVisor を使用するときは 906.9Mbps, そしてネットワーク監視を行う BitVisor を使用する場合は 899.4Mbps であり, 変更を加えていない BitVisor とネットワーク監視を行う BitVisor のオーバーヘッドは, それぞれ 0.78%, 0.94%であった. この結果から BitVisor でパケットフィルタリングを行うことによるオーバーヘッドは十分に小さくゲスト OS のネットワークの性能に大きな影響を及ぼすことはないことが分かる.

5.3 SDN コントローラのよるポートスキャンの検出と通信の遮断

保護ドメインで動作する C 言語で記述した SDN コントローラが Flow Mod によってフローエントリを送信して SDN スイッチで機能するか確認する. 実験では Packet In で SDN コントローラに入ってくるパケットをフィルタリングしてポートスキャンの検出する. そして Flow Mod によって通信の遮断するためのフローエントリを SDN スイッチに挿入する.

実験の概要を次の図 4 に示す. SDN スイッチの Open vSwitch には 2 つの計算機 A, B（表 5）が接続されている.

実験は次に示す 4 つの手順で行い, BitVisor の保護ドメインで動作する SDN コントローラによって A からの攻撃から B を守ることができることを確認した.

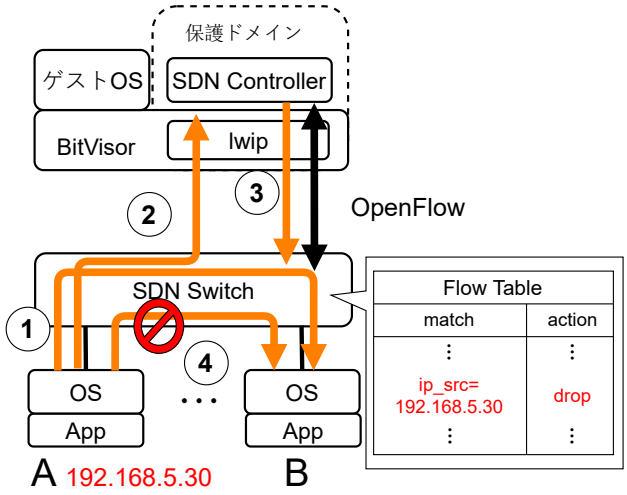

図 4 SDN コントローラのよるポートスキャンの検出と通信の遮断の実験

表 5 実験環境に使用した計算機

	計算機 A	計算機 B
OS	Windows 10	Windows 10
CPU	Intel (R) Core (TM) i5-7200U CPU 2.30GHz	Intel (R) Core (TM) i7-7500U CPU 2.70GHz
RAM	8GByte	8GByte
NIC	Intel Corporation Ethernet Connection 219-V (rev 21)	ASIX AX88772 USB2.0 to Fast Ethernet Adapter

① A が BitVisor 上で動作する OS にポートスキャン
② SDN スイッチに入ってきたフローエントリにマッチしないパケットの処理方法を問い合わせる
③ SDN コントローラでポートスキャンを検出して通信を遮断するためのフローエントリを SDN スイッチに挿入する
④ ポートスキャンを行った A からの通信が遮断される

この実験によって A によるポートスキャンを SDN コントローラで検出して A からの通信を遮断するフローエントリを挿入することができた. これによって A からの攻撃を SDN スイッチで遮断し, B を守ることができた. しかし, ポートスキャンによって通信を遮断するまでに大量のパケットを処理することで通信が遮断されてしまうことがあり通信が不安定となっている. 切断されたコネクションは自動で再接続されるが SDN コントローラの信頼性の向上のために大量のパケットの処理に対応することが課題である.

SDN コントローラが挿入したフローエントリによって通信を遮断できたという確認は A からのポートスキャンの実行前に A から B に TCP パケットを 1 秒毎に送信し,

ポートスキャン実行後に B が A から送信される TCP パケットを受信できるかどうかで確認する．図 4 に示す実験でポートスキャン実行前に A から B に向けて TCP パケットを送信して SDN スイッチのフローエントリに一致したパケットの個数を確認した結果を次の図 5 に示す．

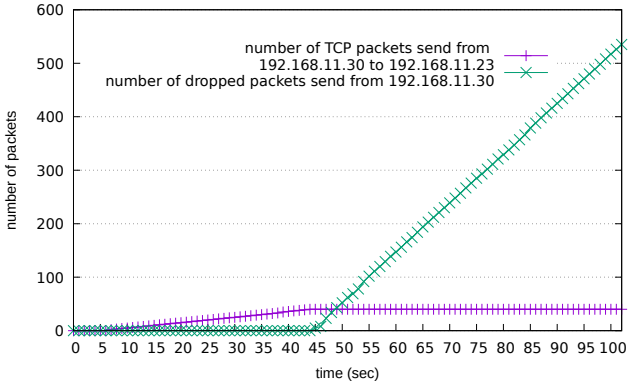

図 5 フローエントリに一致したパケット数

図 5 は SDN スイッチのフローエントリの統計情報から取得した A から B へ TCP パケットを送信するフローエントリに一致したパケットの個数と，A から送信されるパケットをドロップするフローエントリに一致したパケットの個数をグラフにしたものである．

図 5 から計測開始 45 秒あたりで A がポートスキャンを実行していることが分かる．A がポートスキャンを実行したことを BitVisor で動作する SDN コントローラが検出し，A からの通信を遮断するフローエントリを SDN スイッチへ挿入している．このため，A から B へ TCP パケットを送信するフローエントリに一致したパケットの個数は A からの通信を遮断するフローエントリが挿入されたあとは増加せず，代わりに A から送信されるポートスキャンのパケットをドロップするフローエントリに一致したパケットの個数が大きく増加している．

この実験から SDN スイッチに接続された悪意のある計算機がポートスキャンを実行したことを BitVisor の保護ドメインで動作する SDN コントローラで検出してフローエントリを挿入して通信を遮断できるということが確認できた．LVisor に移植された LibVMI によってハイパーバイザからゲスト OS の情報を取得することができればマルウェアに感染したゲスト OS から透過的なネットワークの制御にも役立つ．

6. おわりに

本研究では BitVisor でパケットフィルタリングによってネットワークの制御を行い，保護ドメインで動作する SDN コントローラによってネットワークの制御を行う高セキュアな SDN 基盤を構築した．提案する SDN コントローラはゲスト OS よりも高い特権レベルの VMX root モードの Ring 3 で動作する SDN コントローラである．これによって攻撃層の大きいゲスト OS が危殆化しても，より高い特権レベルで動作する SDN コントローラに悪影響を及ぼすことはなくなる．

今後の課題としては次の 3 つが挙げられる．1 つ目はコントロールプレーンのセキュリティの強化として平分で通信が行われている OpenFlow を暗号化することである．これは BitVisor の主な目的であるクライアント PC のセキュリティ強化のためにディスクやネットワーク I/O の暗号化に使われる暗号化ライブラリを流用することができる．2 つ目は C 言語で記述した SDN コントローラのプログラムを Lua で書き直すことである．SDN コントローラの ELF バイナリは BitVisor のバイナリに埋め込んでいるため，プログラムの変更のたびに BitVisor のバイナリを生成し直す必要があり，柔軟性に欠ける．3 つ目はゲスト OS への通知機構や複数の SDN コントローラが連携し協調して動作することによる耐障害性の向上である．

参考文献

[1] McKeown, Nick, et al. ”OpenFlow: enabling innovation in campus networks.” ACM SIGCOMM Computer Communication Review 38.2 (2008): 69-74.
[2] Dridi, Lobna, and Mohamed Faten Zhani. ”SDN-guard: DoS attacks mitigation in SDN networks.” 2016 5th IEEE International Conference on Cloud Networking (Cloudnet). IEEE, 2016.
[3] Banse, Christian, and Sathyanarayanan Rangarajan. ”A secure northbound interface for sdn applications.” 2015 IEEE Trustcom/BigDataSE/ISPA. Vol. 1. IEEE, 2015.
[4] Tan, Lin, et al. ”Bug characteristics in open source software.” Empirical software engineering 19.6 (2014): 1665-1705.
[5] Chen, Peter M., and Brian D. Noble. ”When virtual is better than real [operating system relocation to virtual machines].” Proceedings eighth workshop on hot topics in operating systems. IEEE, 2001.
[6] Shinagawa, Takahiro, et al. ”Bitvisor: a thin hypervisor for enforcing i/o device security.” Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments. 2009.
[7] Masanori Misono, Kaito Yoshida, Juho Hwang and Takahiro Shinagawa, ”Distributed Denial of Service Attack Prevention at Source Machines”, The University of Tokyo 2018 IEEE 16th Int. Conf. on Dependable Autonomic & Secure Comp.
[8] 野村和裕, 吉村拓也, and 毛利公一. ”仮想計算機モニタを使ったマルウェアの挙動解析.” コンピュータセキュリティシンポジウム 2009 (CSS2009) 論文集 2009 (2011): 1-6.
[9] 松下正吾, et al. ”中立的仮想計算機モニタによる耐タンパーデバイスのアクセラレータの実装.” 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) 2011.9 (2011): 1-8.
[10] 市川遼, and 並木美太郎. ”言語処理系を組み込んだ VMM による OS の効率的な監視システム.” 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) 2017.18 (2017): 1-8.
[11] 市川遼, and 並木美太郎. ”VMM 上で動作するスクリプト言語によるゲスト OS の監視ルール記述.” コンピュータ

システム・シンポジウム論文集 2017 (2017): 12-18.
[12] 市川遼, ”スクリプト言語処理系と VMI を組み込んだプログラマブルな VMM の設計と実装.” 修士論文 2020.1
[13] The Programming Language Lua. http://www.lua.org/ (accessed 2020-01-02)