Vol.2020-OS-148 No.12
2020/2/28

マルチテナント向けコンテナ環境における軽量かつ柔軟なARPスプーフィング対策の実現

中田 裕貴[1,a)]　松原 克弥[1,b)]　松本 亮介[2,c)]

概要：近年，コンテナ型仮想化技術を用いて，マルチテナント向けクラウドコンピューティング基盤を実現する事例が増えている．コンテナ型仮想化システムでは，OS カーネルを共有しつつ，プロセス単位で OS リソースを多重化することで，各コンテナ環境を隔離する．しかし，隔離空間から抜けて，他コンテナに対しての攻撃が可能な OS リソースや機能が存在する．これらの利用は実行権限によって制限されているが，粒度が不十分である．従来，これらの OS リソースの制限をおこなうために，コンテナのセキュリティ機能を用いた制御，仮想マシンを用いた OS 多重化による隔離，ユーザランドにおけるネットワークスタックの再構築などの手法が提案されている．しかし，これらの手法は，設定が柔軟でなかったり，ネットワークのオーバヘッドを増大させる課題があった．本研究では，ハードウェア仮想化技術を用いつつ，OS を多重化せずに軽量かつ柔軟なネットワーク隔離手法を提案する．

1. はじめに

クラウドコンピューティング基盤の中でも，PaaS（Platform as a Service）や FaaS（Function as a Service）と呼ばれるサービス形態では，ユーザが使用する環境の隔離にコンテナ型仮想化技術を使用することが増加している [1], [2], [3]．

コンテナ型仮想化技術は，独立した OS 環境を実現する仮想化技術である．コンテナ型仮想化を実現するソフトウェアであるコンテナランタイムは，Linux Namespaces[4] や Linux Cgroups[5]，chroot[6] などの OS 機能を使用して，ファイルシステムやマウントポイント，プロセス ID などの OS リソースを多重化する．OS リソースを多重化して隔離することで，OS カーネルを共有したままユーザが使用する環境の隔離が実現できる．コンテナ型仮想化技術は，従来のハードウェア仮想化技術に比べて，ユーザが使用する環境を軽量に隔離できる．ハードウェア仮想化技術では，ハイパーバイザがハードウェアリソースを多重化し，仮想的なマシンによってユーザが使用する環境を隔離する．それに対してコンテナ型仮想化技術は，ハードウェアリソースを多重化せず，OS カーネルを共有する．仮想マシンを作成しないことで，ユーザが使用する環境の軽量な隔離を実現している．

コンテナ型仮想化技術には，コンテナに付与するネットワーク権限の粒度に課題がある．コンテナランタイムは，Linux Namespaces と仮想ネットワークインターフェース，Linux Bridge を用いて，仮想的なネットワークデバイスをコンテナに提供する．しかし，ネットワークの隔離空間から抜けて，他コンテナに攻撃などの悪影響を与えることが可能なネットワークリソースが存在する．コンテナランタイムは，これらのネットワークリソースを実行権限によって使用を制限することで，ユーザが使用する環境間が共有されることや他コンテナに対する攻撃を防いでいる．

コンテナ内部で動作するアプリケーションが，制限されたネットワークリソースを使用するためには，特別な実行権限を付与する必要がある．特別な実行権限を付与することで，制限されたネットワークリソースの使用が可能になるが，権限の粒度が不十分であるため，本来必要としないネットワークリソースが使用可能になってしまう．それらのネットワークリソースを用いることで，ARP スプーフィング攻撃といった ARP プロトコルの応答を偽装して別のクライアントになりすます攻撃が実行可能になる．他のユーザのコンテナに対して ARP スプーフィング攻撃を行うことで，コンテナ間通信の盗聴や改変が行われてしまう．

コンテナに付与するネットワーク権限の粒度に関する課題を解決するために，OS カーネル多重化によるネットワークリソース隔離，ユーザランドにおけるネットワーク

[1] 公立はこだて未来大学
Future University Hakodate
[2] さくらインターネット株式会社 さくらインターネット研究所
SAKURA internet Research Center, SAKURA internet Inc., Akasaka, Chuo-ku, Fukuoka 810-0042 Japan
[a)] b1016089@fun.ac.jp
[b)] matsu@fun.ac.jp
[c)] r-matsumoto@sakura.ad.jp

リソース再構築，OS カーネルのセキュリティ機能を用いた制御といった手法がある．これらの手法には，ARP スプーフィングが防止できない，設定が柔軟でない，システムのオーバヘッドを増大させるといった課題がある．

本研究は，ARP スプーフィング攻撃を柔軟かつ軽量に，全てのパケットに介在して防止することを目的とする．目的を達成するために，ハードウェア仮想化技術を用いつつ，OS の多重化をしない軽量なネットワーク隔離手法を提案する．

2. コンテナ型仮想化技術におけるネットワーク隔離の課題

コンテナ型仮想化技術におけるネットワーク隔離の課題として，コンテナに付与するネットワーク権限の粒度がある．コンテナ型仮想化技術は，Linux Namespaces と仮想ネットワークインターフェース，Linux Bridge を用いて，コンテナ毎にネットワークデバイスを提供している．しかし，低レベルネットワークインターフェースや I/O ポート，ネットワーク機能に関連するカーネルモジュールといったネットワークリソースは，ネットワークの隔離から抜けて，他のコンテナやコンテナをホストしている OS に攻撃などの悪影響を与えることが可能である．そのため，隔離から抜けることが可能なネットワークリソースは，コンテナランタイムによって特別な実行権限で制限している．制限されたリソースが必要なアプリケーションの例として，ping がある．ping は，低レベルネットワークインターフェースである RAW ソケットを用いて，ICMP メッセージ[*1]の送受信を行う．そのため，特別な実行権限を付与しない限りコンテナ内で実行できない．

特別な実行権限を付与する方法は，2 種類存在する．1 つ目は，OS が持つ全リソースへアクセスできる root 権限の付与である．root 権限の付与は，ネットワークリソースへのアクセスだけではなく，全ての OS リソースや機能に対する権限が付与されてしまうという課題がある．root 権限の付与によって，共存しているコンテナに影響があるアクセスや制御が可能になってしまう．

2 つ目は，Linux Capabilities[7] を用いた権限の付与である．Linux Capabilities は，root 権限を 37 種類の権限に分割し，組み合わせて権限を付与できる．しかし，Linux Capabilities には，37 種類の分割では粒度が粗いという課題がある．例えば，ping を使用するコンテナを考える．ping を使用するためには，CAP_NET_RAW という権限を付与する必要がある．CAP_NET_RAW を付与することで，OS の TCP/IP プロトコルスタックを迂回して，直接データリンク層とデータ通信を行うことが可能になってしまう．これにより，ARP スプーフィング攻撃が実行可能となってしまう問題がある [8], [9].

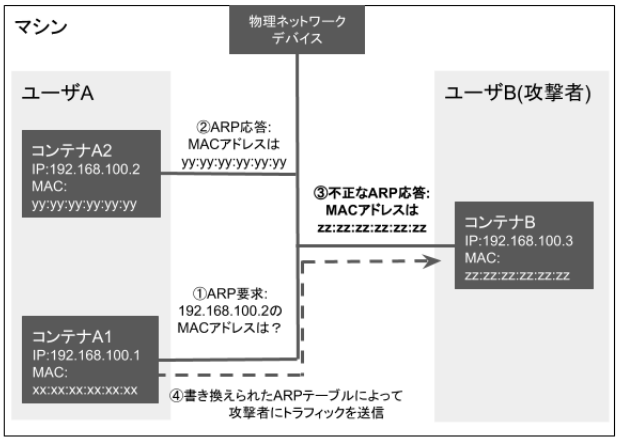

図 1 ARP スプーフィング攻撃の流れ
Fig. 1 The flow of ARP spoofing attack

図 1 は，コンテナ間通信に対しての ARP スプーフィング攻撃の例である．ユーザ A は，コンテナ A1・A2 のコンテナ間で通信を行う．通信を始めるために，コンテナ A1 から ARP 要求をブロードキャストして，IP アドレスから MAC アドレスを解決する．コンテナ A2 は，ブロードキャストされた ARP 要求に対して，正当な ARP 応答を送信して MAC アドレスを返答する．ここで，ユーザ B が不正な ARP 応答をコンテナ A1 に大量に送信する．コンテナ A1 は，正当な ARP 応答ではなく，不正な ARP 応答を ARP テーブルに保存してしまう．その結果，コンテナ A2 ではなく，ユーザ B が持つコンテナ B と通信するようになる．

3. 関連する先行研究

コンテナ型仮想化技術におけるネットワーク隔離の課題を解決するために，仮想マシンによるネットワークリソース多重化，ユーザランドにおけるネットワークリソース再構築，OS カーネルとコンテナのセキュリティ機能を用いる手法がある．

3.1 仮想マシンによるネットワークリソース多重化

仮想マシンによるネットワークリソースの多重化手法として，Kata Containers[10] がある．Kata Containers は，Pod と呼ばれるコンテナ群毎にハードウェア仮想化を用いて仮想マシンを作成する．そして，Pod 単位で別々の仮想マシンとして動作することで，コンテナ群におけるネットワークリソースの多重化を実現している．Kata Containers は，Pod の起動毎に仮想マシンを作成しているため，コンテナの起動にかかる時間が課題である [11]．また，Pod 単位で仮想マシンを作成することで，リソース消費量が増大するという課題もある．

Kata Containers は，仮想マシンを用いてネットワークリソースを多重化することで，ネットワーク隔離から抜け

[*1] 誤り通知や通信に関する情報の通知に使用するメッセージ

て他コンテナやコンテナをホストしている OS への攻撃を防ぐことができている．しかし，コンテナに付与するネットワークに関する権限の粒度が粗い課題を解決していない．また，Kata Containers にはプロトコルや宛先毎などでネットワークアクセスを制御する機能はないため，ARP スプーフィング攻撃を防げない．

3.2 ユーザランドにおけるネットワークリソース再構築

ユーザランドにおけるネットワークリソースの再構築手法として，gVisor[12] がある．gVisor は，Sentry と呼ばれる OS カーネル上に実装されたユーザランドカーネルでコンテナを動作させる．gVisor は Sentry で，ネットワークスタックを再構築している．これにより，ネットワーク隔離を抜けるようなネットワークリソースの隔離を実現している．gVisor は，ネットワークスタックをユーザ空間で再実装しているため，ネットワーク性能にかかるオーバヘッドが課題である [11], [13].

gVisor は，ユーザランドでネットワークリソースを再構築することで，ネットワーク隔離から抜けて他コンテナやコンテナをホストしている OS への攻撃を防ぐことができている．しかし，コンテナに付与するネットワークに関する権限の粒度が粗い課題は解消していない．また，gVisor にはプロトコルや宛先毎などでネットワークアクセスの制御をする機能はないため，ARP スプーフィング攻撃を防げない．

3.3 OS カーネルとコンテナのセキュリティ機能の使用

AppArmor は，プログラム単位でセキュリティポリシに基づいて，強制アクセス制御（MAC:Mandatory Access Control）を実現するためのセキュリティ機構である [14]．強制アクセス制御とは，ファイルの権限設定等は関係なく，強制的にアクセス制限を設けることができる機構である．AppArmor は，セキュリティポリシで RAW ソケットを拒否，ICMP を許可することで，ping コマンドが実行可能な状態のまま，ARP スプーフィング攻撃を防ぐことが可能である．しかし，AppArmor には IP アドレス毎に許可，不許可のような，柔軟な設定はできない．

Calico は，コンテナや仮想マシンに対して L3 ルーティングベースのネットワークキング機能とセキュリティ機構を提供するソフトウェアである [15]．Calico を用いることで，コンテナ群毎にファイアウォールを設定できる．Calico は，セキュリティポリシの設定に，IP アドレス単位やプロトコル単位でのアクセス制御を記述することが可能である．Calico を用いることで，ARP スプーフィング攻撃を防ぐことができる．

Calico のようなセキュリティ機能は，ユーザ空間上で実装されているが，近年コンテナ型仮想化技術と共に，Linux カーネル内で動作する eBPF（extended Berkeley Packet Filter）やカーネル空間をバイパスする DPDK（Data Plane Development Kit）の利用が注目されている．アクセス制御におけるリファレンスモニタという考え方では，改ざんされにくく，全てのアクセスに対して必ず介在可能であり，リファレンスモニタ自身が検証可能であることでアクセス制御を保証する [16]．コンテナで eBPF や DPDK が用いられた場合，OS カーネル上に実装されたセキュリティ機構では，コンテナのパケットに必ず介在することができない．そのため，OS カーネル上で実装されたセキュリティ機能が，リファレンスモニタとして適切にアクセス制御することを難しくしている．

4. ARP スプーフィング攻撃を防止する軽量かつ柔軟なネットワーク隔離の実現

4.1 実現するための要件

本研究では，第 3 章で述べた関連する先行研究の課題から，ARP スプーフィング攻撃を防止できる柔軟かつ軽量なネットワーク隔離機能の実現を目指す．柔軟かつ軽量なネットワーク隔離手法を実現するための必要な要件を以下にまとめる．

(1) カーネル空間内でネットワーク制御を行うソフトウェアやカーネルをバイパスするアーキテクチャにおいても適切なアクセス制御を実現するために，コンテナからのパケット送受信に介在が可能であること
(2) ネットワークリソースの隔離性向上によって，ネットワーク性能が大きく低下するのを防ぐために，ネットワークリソース再構築手法と比べて軽量にネットワーク隔離が実現可能であること
(3) IP アドレス単位での許可・不許可といった柔軟な設定を実現するために，セキュリティポリシを柔軟に記述可能であること
(4) 本研究が対象とする PaaS，FaaS クラウドではコンテナが頻繁に起動と停止を繰り返すため，コンテナが迅速に起動可能であること

4.2 ハードウェア仮想化技術を用いたネットワーク隔離手法の提案

本研究では，ハードウェア仮想化技術を用いたネットワーク隔離手法を提案する．ハードウェア仮想化技術を用いて，パケットを捕捉・制御することで軽量かつ柔軟にネットワークの隔離を実現する．

本提案手法で実現するネットワーク隔離では，ハイパーバイザと仮想ネットワークデバイスを用いる．仮想ネットワークデバイスは，図 2 のように，OS カーネル内の仮想ネットワークではなく，ハイパーバイザに直接パケットを送信する．ハイパーバイザは，仮想ネットワークデバイス経由で受け取ったパケットから，プロセスを解析し，コンテナを特定する．そして，コンテナごとに記述されたセ

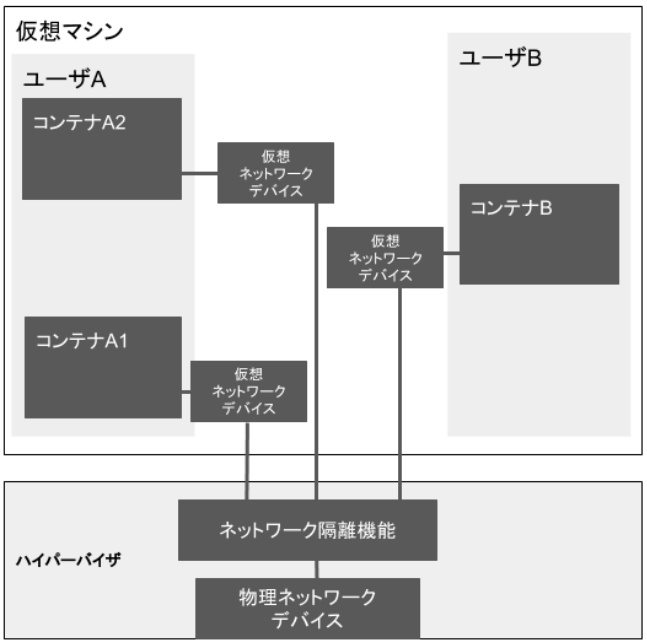

図 2　本提案手法の概要図
Fig. 2　Architecture of the proposed method

キュリティポリシを使用してパケットを検査する．検査の結果，許可された宛先へのパケットであれば，ハイパーバイザは宛先コンテナやマシン外にパケットを送信する．許可されていない宛先へのパケットであった場合，ハイパーバイザはパケットを破棄する．

本提案では，要件（1）と（2）を満たすために，ハードウェア仮想化技術を用いる．要件（1）を満たすためにハードウェア仮想化技術を使用することで，ネットワークデバイスへの全てのパケットを捕捉し，OS 上では捕捉できない全パケットへの介在を実現する．要件（2）を満たすためにハードウェア仮想化技術を使用することで，CPU のハードウェア仮想化支援機能を活用し，gVisor より小さいオーバヘッドを実現する．要件（3）を満たすためにパケットを解析し，IP アドレス等の情報を取得する．IP アドレス等の単位で制御するためのセキュリティポリシには，プログラムとして記述可能な設定ファイルを用いることで柔軟な設定を実現する．要件（4）を満たすために，ハードウェア仮想化技術を用いつつ，OS カーネル全体の多重化は行わない．隔離が必要なネットワークリソースのみ仮想化することで，コンテナの迅速な起動を実現する．

5. ハードウェア仮想化技術を用いたネットワーク隔離手法の実装

ハードウェア仮想化技術を用いたネットワーク隔離は，ハイパーバイザ内に実装する 4 つの機能によって実現する．

第 1 の機能は，OS からのパケット監視機能である．OS からのパケット監視は，全パケットへの介在と，コンテナ毎の柔軟なネットワーク隔離を実現するために行う．

第 2 の機能は，プロセス解析機能である．プロセス解析は，パケットの送信元コンテナを特定するための情報を取得するために行う．

第 3 の機能は，コンテナ特定機能である．コンテナ特定機能は，解析したプロセス情報からコンテナを特定するために用いる．

第 4 の機能は，アクセス制御機能である．アクセス制御機能は，パケットの内容を確認して許可された内容か検査するために用いる．

各機能における技術的課題に対しての実装手順を以下に述べる．

5.1 OS からのパケット監視機能の実装

OS からのパケット監視機能は，コンテナ間で共有するネットワークリソースへのアクセス捕捉が技術的課題である．OS リソース全体の多重化をせずにネットワークリソースの隔離を実現するには，隔離が必要なネットワークリソースだけのアクセスを捕捉しなければならない．

ネットワークリソースのみのアクセス捕捉を実現するために，準パススルーハイパーバイザの BitVisor[17] を用いる．BitVisor はセキュリティ向上が目的のハイパーバイザである．BitVisor の機能を用いることで，ネットワークデバイスなどの I/O を監視し，暗号化やアクセス制御が実現できる．BitVisor が採用している準パススルー型アーキテクチャは，全てのデバイスを仮想化せず，仮想化が必要なデバイスのみを仮想化する．準パススルー型ドライバを用いて必要最低限の I/O のみ補足できるため，コンテナ間で共有するネットワークリソースへのアクセス捕捉という技術的課題を解決できる．

5.2 プロセス解析機能の実装

プロセス解析機能は，BitVisor で取得可能な情報からのプロセス情報取得が技術的課題である．プロセス情報を取得するには，Linux カーネルのプロセスディスクリプタ（task_struct 構造体）を発見する必要がある．BitVisor で取得できる情報は，仮想マシンのレジスタの内容であるため，仮想マシンのレジスタ内容から task_struct 構造体を発見する必要がある．

本実装で用いる Linux カーネルのバージョンは 3.10 とする．プロセスディスクリプタの取得は，カーネルスタックポインタの取得，スレッドディスクリプタの算出，task_struct 構造体からの実行中プロセスに関する情報取得の 3 段階の手順で実現する．

第 1 段階で取得するカーネルスタックは，仮想メモリ空間内のカーネル空間にあるスタック領域である．カーネルスタックを取得することで，現在 CPU で実行中のスレッドの情報を持つ領域を算出できる．BitVisor でのカーネルスタックポインタの取得は，VMCS のゲスト情報領域に保

存されている RSP レジスタを用いる．VMCS とは，Intel VT-x に存在するデータ領域のことである [18]．VMCS のゲスト情報領域には，仮想マシンのレジスタの内容が格納されている．BitVisor 上の仮想マシンからパケットが送信されると，VM Exit というイベントが発生し，制御が仮想マシンから BitVisor に遷移する．遷移した際，RSP レジスタを参照することで，仮想マシンのカーネルスタックポインタを取得できる．

第 2 段階では，スレッドディスクリプタ（threard_info 構造体）の算出によって，現在 CPU で実行中スレッドの情報を取得する．また thread_info 構造体は，task_struct 構造体への参照を持っている．したがって，thread_info 構造体を取得することで，実行中プロセスの task_struct 構造体が取得できる．thread_info 構造体の取得には，第 1 段階で取得したカーネルスタックポインタを用いる．これは，カーネルスタックと thread_info 構造体は共用体として Linux カーネル内で確保されているからである．カーネルスタックポインタからカーネルスタックの先頭アドレスを取得することで，thread_info 構造体の取得が実現できる．

第 3 段階で取得する task_struct 構造体は，プロセス ID やプロセスの状態といったプロセスに関わる情報を一括で格納している．本実装では，コンテナの特定に用いる情報として，プロセスグループ ID を取得する．プロセスグループ ID を用いることで，コンテナランタイムがプロセスを複製して作成したコンテナ内プロセスを特定する．

5.3 コンテナ特定機能の実装

コンテナ特定機能は，セマンティックギャップ[*2]が技術的課題である．コンテナという概念は，OS 上の機能によって実現される．そのため，セマンティックギャップにより解析したプロセス情報だけでは，コンテナを特定することができない．

コンテナを特定するために，コンテナランタイムから，解析したプロセス情報を補完するための情報を BitVisor へ提供する．BitVisor は，提供された情報と，解析したプロセス情報からコンテナを特定する．本実装では，コンテナランタイムとして，コンテナの設定ファイルをプログラムとして記述できる Haconiwa[19] を使用する．Haconiwa は，コンテナの起動や終了などのライフサイクルのイベントを契機とした処理をコンテナ設定ファイル内にプログラムとして記述できる．本実装では，Haconiwa を使用してコンテナ起動時に BitVisor がコンテナ特定に必要な情報を提供する．

Haconiwa から BitVisor へ提供する情報は，Haconiwa が作成したコンテナのプロセスグループ ID である．Haconiwa は，コンテナの作成をする際にコンテナのプロセスグループ ID を取得する．そして，取得したプロセスグループ ID を，OS がハイパーバイザを呼び出す専用命令であるハイパーバイザコールを用いて BitVisor に送信する．

BitVisor でのコンテナ特定は，仮想マシン上からパケットが送信されて VM Exit が発生した際に行う．コンテナの特定には，プロセス解析機能で取得したプロセスグループ ID と，Haconiwa から提供されたプロセスグループ ID を使用する．両方のプロセスグループ ID が一致した場合，そのパケットがコンテナから送信されたパケットであると判断する．

5.4 アクセス制御機能の実装

アクセス制御機能は，パケット解析とコンテナ別アクセス制御が技術的課題である．柔軟な隔離を実現するには，パケットの宛先やプロトコルを解析することが不可欠である．また，コンテナ毎の柔軟なアクセス制御を実現するには，コンテナ毎にセキュリティポリシを用いることが必要である，

パケット解析には，BitVisor の Intel PRO/1000 準パススルードライバを活用する．BitVisor は，ネットワークパケットへの介在を実現するために，パケットのやり取りに使うリングバッファを複製する．リングバッファは，ディスクリプタと呼ばれるパケットが格納されたバッファへのポインタ等を保持している．BitVisor は，複製したリングバッファを用いることで，OS が読み書きするリングバッファと NIC が読み書きするリングバッファを分離する．OS が読み書きに用いるリングバッファを，シャドウ・リングバッファと呼ぶ．BitVisor がこの 2 種類のリングバッファの同期を適切なタイミングで行うことで，ネットワークパケットへの介在を実現している．本機能のパケット解析機能は，リングバッファの同期処理の際にパケットの解析を行うことで，コンテナから送信されたパケットの宛先を特定する．シャドウ・リングバッファのディスクリプタが参照しているバッファには，イーサネット・フレームが格納されている．パケット解析では，イーサネット・フレームの中から ARP パケットを取り出して解析する．ARP パケットでは，32 ビットの宛先 IP アドレス領域を確認する．

コンテナ別のアクセス制御は，セキュリティポリシを Haconiwa のコンテナ設定ファイルの中に記述する．Haconiwa は，コンテナが作成されたのを契機に記述されたセキュリティポリシの内容をハイパーバイザコールを用いて BitVisor へ提供する．BitVisor は，Haconiwa から提供されたセキュリティポリシの内容に基づいて，パケット解析によって特定したコンテナからのパケットの宛先 IP アドレスが，許可された宛先か判断する．許可されていない宛先へのパケットでは，リングバッファからシャドウ・リングバッファへのコピーをスキップしてパケットを破棄する．

[*2] 仮想マシン内部の OS やリソースに関する情報を，ハイパーバイザで取得できる情報から再構築することができないということ．

表 1　実験に用いたマシンの仕様
Table 1　Experimental Environment

OS	CentOS 7
Linux カーネル	3.10（gVisor の評価には 5.4 を使用）
CPU	Intel（R） Core（TM） i5-6260U CPU 1.80GHz
RAM	8GByte
NIC	Intel Coporation Ethernet Connection I219-V （rev 21）

表 2　iperf3 のサーバに用いたマシンの仕様
Table 2　Experiment Environment of iperf3 Server

OS	Ubuntu 18.04.3 LTS
Linux カーネル	4.15
CPU	Intel（R） Core（TM） i7-8550U CPU 1.80GHz
RAM	8GByte
NIC	Intel Corporation Ethernet Connection I219-V （rev 21）

表 3　iperf3 の設定
Table 3　iperf3 Settings

IP バージョン	IPv4
プロトコル	UDP
帯域幅	1000Mbit/sec
通信時間	10 秒

6. 実験と評価

第 4.1 章で述べた，実現のための要件（2）を評価するためにネットワークのオーバヘッドを計測した．また，実現のための要件（4）を評価するためにコンテナ起動時間を計測した．

6.1 ネットワークにかかるオーバヘッドの実験と評価

6.1.1 実験概要

本実験は，ネットワークにかかるオーバヘッドを評価するために iperf3 でスループットを測定する．

評価対象は，仮想化なし，runC（プロセス分離コンテナ），Kata Containers，gVisor，Haconiwa，ネットワーク仮想化を無効にした BitVisor，ネットワーク仮想化を有効にした BitVisor，ネットワーク仮想化・プロセス解析機能を有効にした BitVisor，Haconiwa とネットワーク仮想化・プロセス解析機能を有効にした BitVisor（本提案手法）である．評価は，データの送信を行うクライアントと，データを受信するサーバの 2 台のマシンを用いてスループットを 10 回計測した際の平均値を用いる．クライアントの仕様を表 1，サーバの仕様を表 2，iperf3 の設定を表 3 に示す．

6.1.2 実験結果と考察

実験結果を表 4 に示す．

仮想化なしのスループットに対して，ネットワーク仮想化を無効にした BitVisor のスループットは大きな差がなかった．ネットワーク仮想化を無効化した BitVisor は，ネットワークをパススルーしているためである．また，ネットワーク仮想化を有効にした BitVisor の場合は，スループットは 1.83%低下する．

表 4　スループットの計測結果
Table 4　The Result of Throughput

	スループット（Mbps）
仮想化なし	941.7
BitVisor（ネットワーク仮想化無効）	939.5
BitVisor（ネットワーク仮想化有効）	924.4
BitVisor（ネットワーク仮想化・プロセス解析機能有効）	773.4
Haconiwa + BitVisor（ネットワーク仮想化・プロセス解析機能有効）	770.7
runC	942.1
Kata Containers	944.9
gVisor	90.9
Haconiwa	940.0

次に，ネットワーク仮想化と，プロセス解析機能を有効化した BitVisor でのスループットは，スループットはプロセス解析機能有効化では 16.33%，ネットワーク仮想化機能も有効化の時は 17.87%低下した．プロセス解析機能は，解析の際にパケットごとに OS が参照している物理アドレスを，BitVisor のアドレス空間にマップする．そして，解析が終了するとアンマップする．そのため，メモリ空間のマップ・アンマップがスループットの低下を招いている．提案手法とネットワーク仮想化とプロセス解析機能を有効化した BitVisor ではスループットに大きな差が無かった．このため，プロセス解析機能がオーバヘッドの多くを占めていることを示している．

runC と Kata Containers，Haconiwa のスループットはそれぞれ，仮想化なしとのスループットに大きな差がなかった．これは，runC と Haconiwa はネットワークはホストマシンのインターフェースを用いているからである．Kata Containers は，仮想マシンを作成しているが，仮想化なしのスループットと大きな差がなかった．Kata Containers のスループットが仮想化なしと変わらない理由は分かっていないため，今後調査する必要がある．gVisor は，スループットが大幅に低下している．これは，ユーザ空間にネットワークスタックを再実装した際のオーバヘッドと考える．

この結果から，本提案手法のネットワーク隔離手法は，17.87%のネットワーク性能の低下を引き起こす．しかし，gVisor と比較すると，8.4 倍のスループットを達成し，実現のための要件（2）を満たせた．

6.2 コンテナの起動にかかる時間の実験と評価

6.2.1 実験概要

本実験では，本提案手法がコンテナの起動にかかる時間に与える影響を評価する．

表 5　コンテナの起動にかかる時間の計測結果
Table 5　The Result of Container Startup Time

	起動にかかる時間（秒）
runC	0.16
Kata Containers	1.53
gVisor	0.21
Haconiwa	0.11
提案手法（Haconiwa）	0.16

評価対象は，runC，Kata Containers，gVisor，Haconiwa，そして提案手法である．実験内容は，コンテナを起動して標準出力に Hello World と出力する．このコンテナの起動から終了までの時間を計測する．評価には，10 回計測した値の平均を用いる．

6.2.2　実験結果と評価

実験結果を表 5 に示す．実験の結果，runC と gVisor，Haconiwa，提案手法の結果から，提案手法とそれ以外のプロセス分離コンテナはコンテナの起動にかかる時間の差は小さい．また，Haconiwa と提案手法のコンテナの起動にかかる時間の差から，本提案手法がコンテナの起動にかかる時間に与える影響は大きくないと判断する．Kata Containers は，起動にかかる時間が，runC と比較して大きい．これは，仮想マシンの作成と OS の起動によるものだと考察する．

これらの結果から，本提案のネットワーク隔離手法が，コンテナの起動にかかる時間に与える影響は小さいと結論付ける．また，仮想マシンによる OS 多重化を行なっている Kata Containers と比較すると，コンテナの起動にかかる時間は約 9 分の 1 であった．

7.　まとめ

本研究は，コンテナに付与する権限粒度の課題によって可能となる ARP スプーフィング攻撃の防止を目的とする．目的達成のために，OS カーネル上では介在できないパケットも制御可能な柔軟かつ軽量なコンテナ向けネットワーク隔離機能を提案した．この提案を実現するために，OS からのパケット監視・プロセス解析機能，コンテナ特定機能，アクセス制御機能を実装した．

本提案手法の有効性を確認するために，ネットワークにかかるオーバヘッドとコンテナの起動にかかる時間の実験と評価を行なった．ネットワークにかかるオーバヘッドは，実現のための要件（２）を満たせたが，プロセス解析機能が大きなボトルネックであることが判明した．コンテナの起動にかかる時間は，既存のプロセス分離コンテナと比較して，大きな差は無く，実現のための要件（4）を満たせた．

今後の課題は 2 つ存在する．第 1 の課題は，コンテナ間通信アクセス制御の実装である．これは，現在の実装ではコンテナから外部マシンのパケットしか補足できないためである．そのため，コンテナ間通信のアクセス制御手法を検討し，確立する．第 2 の課題は．プロセス解析機能の高速化である．第 6.1.2 項で示したように，プロセス解析機能の影響によってネットワーク性能が大きく低下している．オーバヘッドを解消するために．プロセス解析機能の高速化と実装を行う．

参考文献

[1] Google LLC: App Engine, `https://cloud.google.com/appengine/` (参照 2019-01-21).
[2] Alibaba Cloud: Elastic Container Instance, `https://www.alibabacloud.com/products/elastic-container-instance` (参照 2019-01-21).
[3] GMO ペパボ株式会社：ロリポップ！マネージドクラウド，`https://mc.lolipop.jp/` (参照 2019-01-21)).
[4] Michael, K.: namespaces(7), `http://man7.org/linux/man-pages/man7/namespaces.7.html` (参照 2019-01-27).
[5] Michael, K.: cgroups(7), `http://man7.org/linux/man-pages/man7/cgroups.7.html` (参照 2019-01-27).
[6] Michael, K.: chroot(2), `http://man7.org/linux/man-pages/man2/chroot.2.html` (参照 2019-01-27).
[7] JM Project: CAPABILITIES, `https://linuxjm.osdn.jp/html/LDP_man-pages/man7/capabilities.7.html` (参照 2019-01-21).
[8] Hertz, J.: Abusing privileged and unprivileged linux containers, *Whitepaper, NCC Group*, Vol. 48 (2016).
[9] Liz, R.: KubeCon + CloudNativeCon North America 2019 : CAP_NET_RAW & ARP Spoofing in Your Cluster, `https://static.sched.com/hosted_files/kccncna19/72/ARP%20DNS%20spoof.pdf` (参照 2019-01-21).
[10] The OpenStack Foundation: Kata Containers: The speed of containers, the security of VMs, `https://katacontainers.io` (参照 2019-01-21).
[11] Xu, W.: KubeCon + CloudNativeCon North America 2018 : Kata and gVisor : A Quantitative Comparison, `https://static.sched.com/hosted_files/kccna18/39/kata-containers-and-gvisor-a-quantitave-comparison.pdf` (参照 2019-01-21).
[12] google LLC: gvisor: container runtime sandbox, `https://github.com/google/gvisor` (参照 2019-01-21).
[13] Young, E. G., Zhu, P., Caraza-Harter, T., Arpaci-Dusseau, A. C. and Arpaci-Dusseau, R. H.: The true cost of containing: A gVisor case study, *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)* (2019).
[14] Ubuntu Documentation Team: AppArmor, `https://wiki.ubuntu.com/AppArmor` (参照 2019-01-21).
[15] Tigera, Inc.: Project Calico - Secure networking for the cloud native era, `https://www.projectcalico.org/` (参照 2019-01-21).
[16] 海外浩平：Linux のセキュリティ機能 : 1. OS へのセキュリティ脅威と Linux の強制アクセス制御，情報処理，Vol. 51, No. 10, pp. 1249–1256 (2010).
[17] Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E. and 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, Association for Computing Machinery, pp. 121–130 (2009).
[18] Uhlig, R., Neiger, G., Rodgers, D., Santoni, A. L., Martins, F. C. M., Anderson, A. V., Bennett, S. M., Kagi, A., Leung, F. H. and Smith, L.: Intel virtualization technology, *Computer*, Vol. 38, No. 5, pp. 48–56 (2005).
[19] 近藤宇智朗，松本亮介，栗林健太郎：Haconiwa: プログラムによる, 組み立て可能性と拡張性を持つ Linux コンテナ，第 80 回全国大会講演論文集，Vol. 2018, No. 1, pp. 19–20 (2018).