コンテナの脆弱性に関するセキュリティ演習環境の構築

西村 拓也[1] 白石 善明[1,2] 小津 喬[2] 橋本 真幸[3]
毛利 公美[4] 葛野 弘樹[1] 森井 昌克[1]

概要： コンテナ型仮想化技術は普及期に入っており，今後さらにコンテナの利用が進むことが予想される．コンテナを導入し安全に運用するためにはコンテナに関するセキュリティの知識を持つ人材の育成が重要となる．コンテナ型仮想化による演習環境でコンテナの脆弱性に関する演習を行う場合，演習システム自体が攻撃対象となるため，支障なく演習システムを運用することが課題である．本論文ではコンテナに関する演習をするためのコンテナとハイパーバイザを併用する演習環境を提案している．また，コンテナのセキュリティを網羅的に学習するための演習シナリオ作成の考え方について示し，演習例を示している．

Building a Security Exercise Environment Regarding Container Vulnerabilities

TAKUYA NISHIMURA[1] YOSHIAKI SHIRAISHI[1,2] TAKASHI OZU[2] MASAYUKI HASHIMOTO[3]
MASAMI MOHRI[4] HIROKI KUZUNO[1] MASAKATU MORII[1]

1. はじめに

コンテナ技術の普及に伴い，コンテナを安全に利用することの重要性が高まっている．コンテナ型仮想化は他の仮想化方式と比較して分離レベルが低く，セキュリティ上の問題が多いとされている．

企業や組織などでコンテナを安全に導入するためにはコンテナのセキュリティに関する知識を持つ人材の獲得，育成が重要である．コンテナに関する演習シナリオを作成する際，演習参加者がコンテナセキュリティを網羅的に学習することができるような演習内容を構成することが求められる．同時に，短期的な学習で幅広い知識を身に着けるために，それぞれの演習から得られる知見に重複が少ない演習メニューを構成することが望まれる．本論文では体系的なコンテナに関するセキュリティ演習を可能とする環境構築を目的としている．

演習環境や Capture The Flag（CTF）などの大会におけるインフラ構築において，コンテナ型仮想化がメモリ効率の観点から注目されている[1], [2]．しかし，コンテナがコンテナホストを攻撃するような（コンテナブレイク）演習を行う場合，コンテナ型仮想化による演習環境では，演習システム自体が攻撃対象となるため，演習システムを支障なく運用することが課題となる．

本論文ではコンテナセキュリティを網羅的に学習するための演習シナリオ作成の考え方について述べる．そしてハイパーバイザとコンテナを併用して，コンテナに関する演習を支障なく行うことができる環境を構築し，演習例を示す．本論文ではハイパーバイザ型仮想化として OpenStack を，コンテナ型仮想化として Docker を用いている．

2. コンテナ技術の普及と運用時の課題

情報技術の発展に伴いシステムの運用管理が複雑化しており，管理コストの軽減などを目的として仮想化技術が用いられるようになってきている．柔軟性，軽量な動作，可搬性などの特徴を持つコンテナ型仮想化が注目を集めている．コンテナ型仮想化とは OS 上に仮想的に独立した領域を作り出し，その中でアプリケーションを動作させる仕組みである．

コンテナ型仮想化の利用には多くのメリットがある一方で，運用には課題があるとされている．国内の企業 420 社に対するコンテナ導入状況の調査[3]によると，「コンテナを導入する計画／検討がある」，「導入構築／テスト／検証段階にある」企業が 39.7%であった．また，導入時に課題となる点として，「問題発生時に対応」が 32.5%，「セキュリティ対策」が 30.2%であった．このようにコンテナ技術を安全に運用するためには障害発生時の対応力やコンテナセキュリティに関する知識をもつ人材が必要となる．コ

1 神戸大学大学院工学研究科
Graduate School of Engineering, Kobe University
2 国際電気通信基礎技術研究所 先端セキュリティ研究室
Department of Advanced Security, ATR
3 株式会社 KDDI 総合研究所
KDDI Research, Inc.
4 近畿大学情報学部
Faculty of Informatics, Kindai University

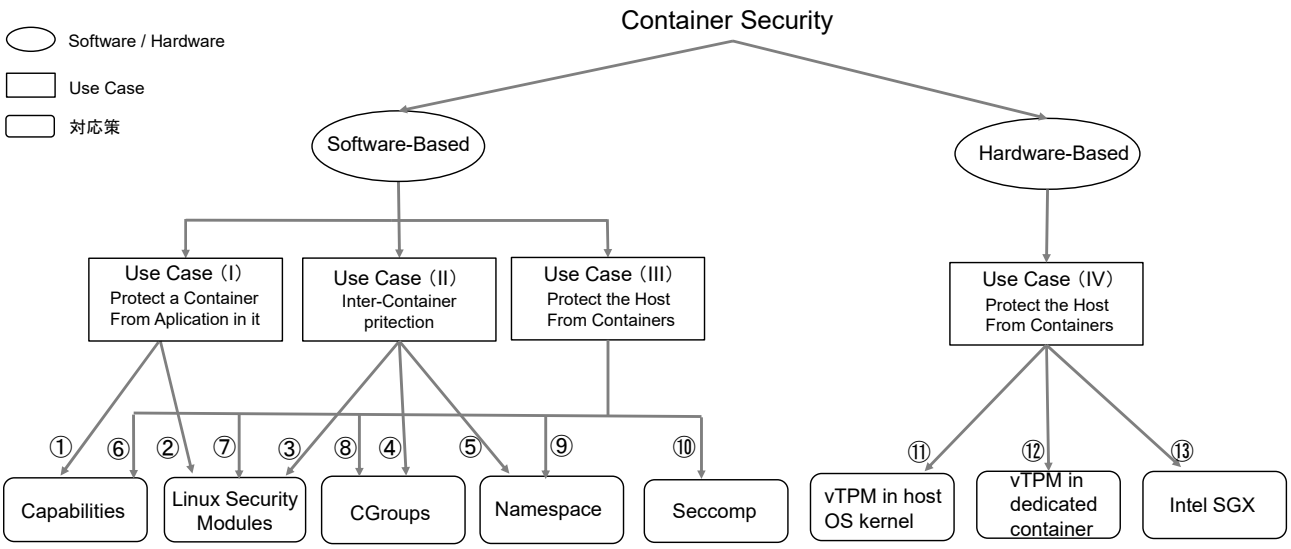

図 1　コンテナセキュリティに関する攻撃の分類と攻撃の影響を軽減すると考えられる対応策（文献[9]をもとに作成）

ンテナの普及に対してエンジニアの不足が予想されるため，コンテナに関する知識を持つ人材の獲得，育成が重要であると結論付けられている．

コンテナを標的としたマルウェア“kinsing”が報告[4]されているように，コンテナの普及に伴い，今後コンテナを標的としたサイバー攻撃の増加が予想される．すなわち，コンテナ技術を安全に運用するうえで，コンテナセキュリティに関する学習を行う必要性が高まっている．そこで本論文ではコンテナセキュリティに関するサイバー演習を行うことができる環境の構築を目的とする．

3. 関連研究および本論文でのアプローチ

3.1 コンテナを用いたサイバー演習環境

サイバー演習とは情報システムにおけるインシデントへの対応手順やその原因を学習するための演習であり，演習管理者が用意したインシデント発生を想定したシナリオに基づき学習を進める．セキュリティ人材育成のためサイバー演習に関する研究が進められている[5], [6], [7].

メモリ効率の良さの観点からサイバー演習や Capture The Flag（CTF）においてもコンテナを用いた研究[1], [2]が進められているが，演習内容に制限がある[8]．コンテナ型仮想化ではカーネルなどをコンテナホストと共有するため分離レベルが低く，あるコンテナでの影響がシステム全体に影響が及ぶ可能性がある．分離レベルとは同じハードウェア上で動作する他の仮想マシンからの独立性を表す．また他の仮想化方式と比較してコンテナ上のプロセスからのハードウェアへのアクセスには差異が生じる可能性がある．

実際にコンテナを攻撃する演習を行う場合，演習管理サーバ自体を攻撃対象とするため演習の運用には注意を要す．次に示すような演習は演習内容に組み込む際に注意が必要であるとされている．

・共有するリソースを上限まで利用するプログラム
・制限されている物理デバイスへアクセス，特に変更を伴うプログラム
・コンテナエンジン等の仮想化の仕組みを悪用したプログラム

3.2 コンテナのセキュリティ

文献[9]で，ホスト・コンテナへの攻撃を次の 4 つのユースケースに分類され，それぞれの攻撃への対策方法を図 1 のようにまとめられている．（I）コンテナ内のアプリケーションからコンテナを保護する，（II）コンテナ間の保護，（III）コンテナからホストを保護する，（IV）悪意のあるまたは不誠実なホストからコンテナを保護する．

コンテナに関する攻撃は Software-Based と Hardware-Based に分類され，さらに Use Case（I）～（IV）に分類される．それぞれの Use Case に対し，被害を軽減すると考えられる対応策が分類されている．

（I）の攻撃例として，脆弱性が含まれているソフトウェアがインストールされているイメージを配布するなどが挙げられる．例えば，脆弱性を持たせた Web サーバの Docker イメージなどを誤って使用した場合，攻撃者は容易にコンテナに侵入することができる．（II）の攻撃例として，ARP spoofing を行うなど Docker ネットワークを通じてあるコンテナが他のコンテナを攻撃するなどが挙げられる．（III）の攻撃例は，コンテナがコンテナホストを攻撃（コンテナブレイク）するような場合である．CVE-2019-5736 ではコンテナホストの root 権限を取得することができる．（IV）ではホストからコンテナ内のデータに不正にアクセスするなどが挙げられる．サイドチャネル攻撃などが例である．

次に図 1 に記載のそれぞれの対応策について説明する．

Capability：管理者権限等を細分化して付与する機能，コンテナへの権限付与について設定を行うことができる．

Linux Security Modules（LSM）：Linux において，様々なセキュリティモデルをサポートするための Linux カーネルのセキュリティフレームワークである．本論文では特に LSM 上に実装される強制アクセス制御機構である AppArmor を用いる．

CGroup：ユーザプロセスをグループ化し，リソース（CPU、メモリ、ディスク I/O など)の利用を制限・隔離する Linux カーネルの機能であり，コンテナ内で利用可能なリソースを制御する．

Namespace（名前空間）：専用の名前空間をプロセスに対して提供し，プロセスに対して分離されたグローバルリソースを提供する仕組みである．

Seccomp（secure computing mode）：システムコール呼出し可否を制御する Linux カーネルのセキュリティ機能，コンテナ内で利用するシステムコールのフィルタリングを行うことができる．

TPM（Trusted Platform Modules）：コンピュータのマザーボードに直付けされているセキュリティに関する各種機能を備えた半導体部品で，データの暗号化・復号化や鍵ペアの生成，ハッシュ値の計算，デジタル署名の生成・検証などの機能を有する．vTPM は仮想マシンなどから TPM を利用するための機能である．

vTPM in host OS kernel：vTPM をホスト OS のカーネルに配置する．コンテナに vTPM を割り当てるには、コンテナ・マネージャがホスト OS カーネルに新しい vTPM を作成するように要求し，それを新しいコンテナに割り当てる．

vTPM in dedicated container：vTPM を信頼できるコンテナに配置し，ほかのコンテナからはそのコンテナを通して vTPM の機能を利用する．そのために専用のソフトウェアアダプタが必要となる

Intel SGX：メモリ上に暗号化かつ隔離された実行領域を生成し，プログラムを実行する Intel CPU の拡張機能である．

本論文では図 1 で Software-Based に分類される攻撃への対策について網羅的に演習することを目的とする．つまり図 1 中の Software-Based 以下の部分グラフのすべての経路を通るような演習メニューの作成し，演習例として示す．

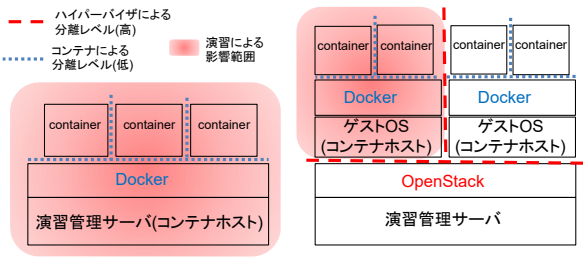

図 2　コンテナブレイクなどの演習を行った場合の影響範囲

3.3 本論文でのアプローチ

コンテナの脆弱性に関する演習を行う場合，演習環境にコンテナ型仮想化が実装されている必要がある．しかしながら，3.1 節に述べたように，サイバー演習の環境構築にコンテナを用いる場合，コンテナの脆弱性に関する演習は，演習システム自体に影響が及ぶ可能性があり運用が困難であるとされている．本論文ではコンテナの利用によって運用コストを軽減しつつ，図 2 のようにコンテナとハイパーバイザを併用することによってこれらの演習を支障なく運用できるシステムの構築手法を提案する．

なお，ハイパーバイザ型仮想化ではなくホスト型仮想化を利用することによってもコンテナの脆弱性に関する演習は実現可能であるが，本論文の環境では多くの仮想マシンの作成・削除に対応可能なように，文献[10]にあるより起動にかかる時間の少ないハイパーバイザ型仮想化を用いる．

4. 演習シナリオ作成

4.1 網羅的な演習メニュー

文献[9]ではホスト・コンテナのセキュリティ要件を満たすと考えられる脅威に対する対応策が図 1 のようにまとめられている．

図 1 のグラフの同じ経路を通る演習メニューを複数作成した場合，演習参加者は重複した内容を学習することになる．重複なく網羅的に学習できるような演習メニューを作成することができれば演習参加者は効率的にコンテナセキュリティに関する知識を身に着けられる．

本論文では図 1 の Software-Based に該当するグラフの葉にある対応策①～⑩を網羅的に演習することを目的とした演習メニューの作成する．また同じ対応策であっても経路が違う場合，そこから学ぶことができる内容は異なると考えられるため別の番号を割り振っている．

4.2 それぞれのユースケースにおける学習難易度の違い

本論文ではコンテナの脆弱性は攻撃者がコンテナまた

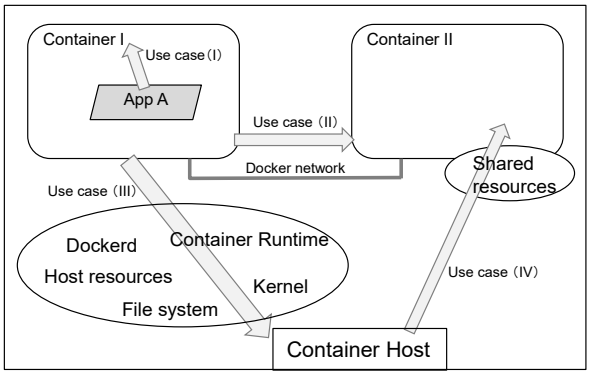

図 3　それぞれのユースケースにおける攻撃形態の違い

はコンテナホストに侵入を許した後の状況を想定している．コンテナセキュリティに関する演習を行う場合，図 3 のようにユースケース（I）～（IV）の攻撃形態の違いから学習難易度が異なると考えられるため注意が必要である．

ユースケース（I）では攻撃による影響範囲がコンテナ内のみに収まる場合が多く，1 つのマシン内での脆弱性のように考えることができるため学習難易度は低い．（II）では Docker ネットワークを通じた攻撃が多く，二台のマシン間のように考えることができるため比較的学習難易度は低い．（III）ではコンテナランタイム，カーネルなどの脆弱性を利用した攻撃が多く，コンテナとホストの関係性を理解しなければならない場合が多い．このような攻撃はコンテナ特有の性質によるものであり，学習難易度は高いと考えられる．（IV）ではハードウェアでの攻撃に分類される．コンテナと共有するハードウェアに対してアクセスするためこれもコンテナ特有の性質によるものである．またハードウェアを利用した攻撃は一般的に学習難易度が高いと考えられる．

5. 演習環境

本論文では 4 章でまとめた方針に沿うコンテナセキュリティに関する演習内容について，コンテナとハイパーバイザを併用することによって支障なく運用する手法を提案する．本論文ではコンテナ型仮想化に Docker，ハイパーバイザ型仮想化に OpenStack を用いた．

5.1 OpenStack，Docker

OpenStack はプールされた仮想リソースを使用してプライベートクラウドやパブリッククラウドを構築および管理するプラットフォームであり[11]，インフラ構築に仮想化を用いる場合に比較的よく利用されるツールである[12].

Docker とはコンテナ化されたアプリケーションを構築，デプロイ，管理するためのオープンソース・プラットフォームであり[13]，柔軟性，軽量な動作，可搬性などの特徴を持つ．

5.2 演習全体の流れ

演習は図 4 のような流れで行う．演習参加者は演習用に用意された Web 画面の説明に従って演習を進める．図 5 のように，参加者は用意されている演習一覧の中から演習を選択し，演習内容に関する説明を読んだうえで，仮想マシン，ネットワークの作成を Web 画面から簡単な操作によって行う. 参加者は作成した仮想マシンに割り振られた IP アドレスの情報を Web 画面から得られる．仮想マシンには SSH 接続し，Web 画面に記載されている脆弱性の説明や攻撃方法を読み，脆弱性の内容を理解したうえで攻撃の操作を行うことによってセキュリティへの理解を深める．

Docker のバージョン等によって再現可能な脆弱性が異なるため，1 台のマシンに脆弱性を集約することは困難である．そこで，あらかじめマシンのイメージを用意しておき，参加者の選択に応じて動的に演習環境を作成する手法が適当である．

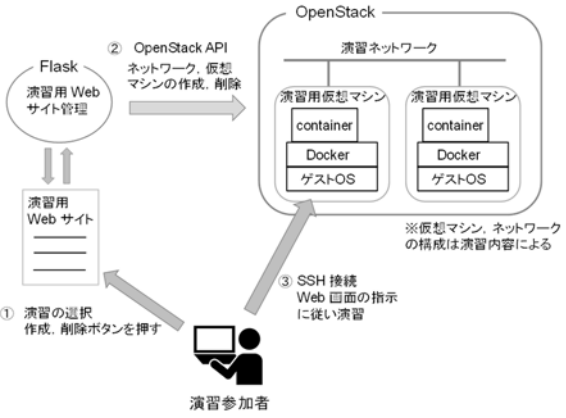

図 4　演習全体の流れ

図 5　演習参加者の Web 画面．演習教材を右側フレーム内の一覧から選択

5.3 演習用仮想マシンの作成

本論文では外部ネットワークへの接続のない演習ネットワークを利用するため，演習に必要なソフトウェアや攻撃ファイルをあらかじめすべて準備した仮想マシンのイメージを用意する．本論文では演習用仮想マシンのイメージ作成に VirtualBox を用いた．VirtualBox とは Oracle 社が開発するオープンソースの仮想化ソフトウェアである[14].

VirtualBox に演習に使う OS をインストールし，演習に用いるアプリケーションなどをすべてインストールしたうえで，起動時のディスクパーティション設定およびルートパーティションのサイズ，MAC アドレス情報，SSH サーバの設定などを行い，演習用イメージを作成した．なお本論文ではこれらの一部の設定を Cloud-Init を用いることによって効率的に行った. Cloud-Init とは仮想マシンの初期設定を自動化するためのツールである．

コンテナセキュリティの演習を行う場合，演習に必要なツールを 1 つのイメージにまとめることができる場合が多い．一度作成した演習用イメージを他者と共有することによって共同開発が容易となる．

6. 演習例

本論文では演習例として，図 1 の software-based に該当する①～⑩についての演習の例を示す．つまり，すべての経路を通るような演習の組を示す．本論文で示す演習内容は表 1 のとおりである．

表 1　本論文で行う演習内容と図 1 対応策との関係

演習内容	図 1 対応策
悪意のあるイメージの利用（SSH）	②
コンテナ内で reboot の実行	①
コンテナ間の ARP spoofing	③,⑤
コンテナ内で大量のリソース消費	④,⑧
runC の脆弱性（CVE02019-5736）	⑦,⑨,⑩
特権コンテナの悪用	⑥

コンテナ間の ARP spoofing（③，⑤），コンテナ内で大量のリソース消費（④，⑧），runC の脆弱性（⑦，⑨，⑩），特権コンテナの利用（⑥）の演習は他のコンテナや演習環境自体に影響を及ぼすため，コンテナ型仮想化のみを用いる演習環境では運用が困難な演習である．

悪意のあるイメージの利用（②），コンテナ内での reboot の実行（①）の演習では対応策の学習のために，演習参加者が docker グループに属するユーザを操作する必要があり，コンテナ型仮想化のみを用いる演習環境では運用に注意が必要である．

コンテナを安全に利用するためのもっともよい方法は最新バージョンのカーネルや Docker を利用することであるが，0-day 攻撃などへの対応のため，様々なセキュリティ対策の手段を知っておく必要がある．

6.1 悪意のあるイメージの利用

コンテナ型仮想化を利用するメリットのひとつにイメージの利用があげられる．他者の作成したイメージの利用は便利である反面，どのように作成されたのかが不明瞭である．本節では Docker イメージに SSH サーバがインストールされていることを公表していないという場合を想定している．これは Use Case（I）に該当する．

何も対処せずにコンテナを起動した場合，コンテナホストから（ポートフォーワディングした場合外部から）SSH 接続することができる．これに対する対応策として AppArmor の利用が挙げられる．対応の様子を示した図 6 のように，AppArmor を用いてコンテナ内で.ssh ディレクトリへのアクセスを禁止すればコンテナへの SSH 接続ができなくなる．このようにコンテナ内で使用しないディレクトリへのアクセスを禁止しておくことが重要である．

図 6　AppArmor によるコンテナへの SSH 禁止（悪意のあるイメージの利用（②））

6.2 コンテナ内で reboot の実行

通常コンテナ内では reboot など systemd を使用することはできない．しかしコンテナに通常以上の権限を付与（--privileged オプションなどを付与）したうえで起動すれば，このようなコマンドの実行が可能となる．本節ではコンテナ内で reboot を行う．その結果，図 7 のようにコンテナ上で再起動の処理を適切に行うことができず，docker ps コマンドの出力結果からもわかるようにコンテナが消える．これは Use Case（I）に該当する

これは Capability の設定を誤ってコンテナに通常以上の権限を付与したことが原因である．このようなことからコンテナを安全に利用するには，Capability の設定を適切に用いて，必要最低限の権限でコンテナを操作することが重要である．

図 7　コンテナ内で reboot を実行したことによるコンテナの終了（①）

6.3 コンテナ間の ARP spoofing

あるコンテナからほかのコンテナを攻撃する方法に ARP spoofing がある．同じネットワーク内に存在する他のコンテナに対して ARP spoofing を行うことによって図 8 の

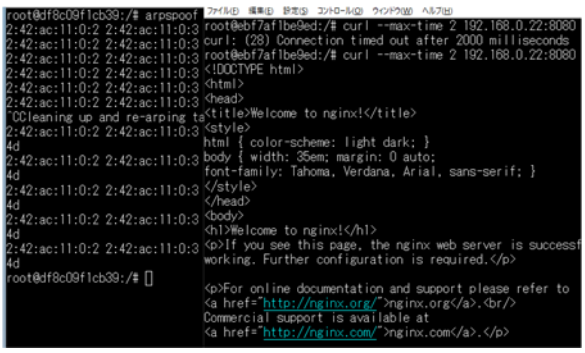

図 8　コンテナ間の ARP spoofing

図 9　AppArmor による ARP spoofing で利用される raw socket の使用禁止（③）

ように攻撃対象コンテナは名前解決を行うことができなくなる．これは Use Case（II）に該当する．

これに対する対応策として，(i) 新たに Docker ネットワークを作成し，それぞれのコンテナを別のネットワークに配置する，(ii) AppAromor を用いてコンテナ内での raw socket の使用を禁止する，が挙げられる．

(i) Docker ネットワークを新たに作成しコンテナを別のネットワークに配置することで，コンテナ間は通信できなくなり ARP spoofing を防ぐことができる．Docker ネットワークの作成には namespace に関する知識を必要とする．コンテナを同じネットワークに配置する必要がない場合はネットワークを分けることでセキュリティの向上につながる．

(ii) ARP spoofing は raw socket を使用しているため，AppArmor を用いてコンテナ内での raw socket の使用を禁止することによって解決できる．raw socket の使用が制限されたコンテナ内で ARP spoofing を行っても，図 9 のように ARP spoofing が実行できなくなっている．このように必要のない機能の使用を制限しておくことがセキュリティの向上につながる．

6.4 コンテナ内で大量のリソース消費

コンテナ型仮想化ではカーネルやメモリをコンテナホストと共有しているため，あるコンテナでメモリを大量に消費した場合，図 10 のようにコンテナホストや他のコンテナが利用可能なリソースが枯渇する場合がある．これは Use Case（I），(II) に該当する．コンテナを標的としたマルウェア“kinsing”[4]はこの性質を利用したものである．

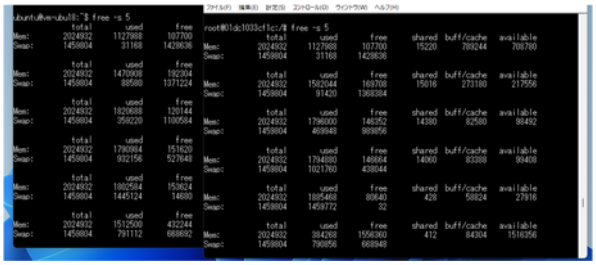

図 10　コンテナ内で大量のリソース消費（左：コンテナホスト，右：コンテナ）

Docker ではこのような問題を解決するため，CGroup を用いてコンテナ内で利用可能なリソースを制限する機能が用意されている．図 11 ではコンテナ内で利用可能なメモリを 512MB に制限している．このようにコンテナで利用するリソースを制限することでセキュリティの向上につながる．

6.5 runC の脆弱性（CVE-2019-5736）

CVE-2019-5736 では Docker コンテナ等で使用する runC が意図せず上書きされ，図 12 のようにコンテナホスト上で root 権限でコマンドが実行される（コンテナブレイクアウト）という脆弱性が存在する．コンテナ内の/proc/pid/exe がホストのバイナリを指すシンボリックリンクなっており，

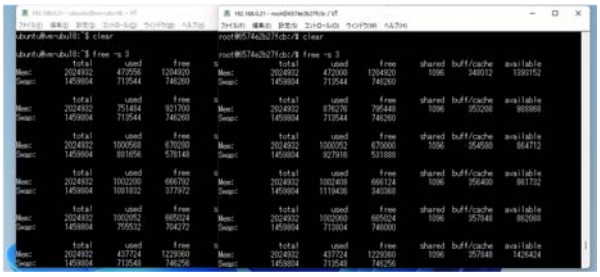

図 11　CGroup によるリソースの制限（512MB）（④，⑧）

攻撃者はこれを用いてコンテナ内部からホストのバイナリを書き換える．図 12 ではコンテナホスト root のみが閲覧できるファイルを閲覧している．攻撃の再現には（https://github.com/twistlock/RunC-CVE-2019-5736）を使用した．これは Use Case（III）に該当する．

図 12　runC の脆弱性（CVE-2019-5736）exploit

この脆弱性対応策として，(i) コンテナのユーザ名前空間にコンテナホストの非 root ユーザをマッピングする．(ii) Apparmor によるアクセス制御．(iii) Seccomp を用いてシステムコールをフィルタリングする．が挙げられる．

(i) コンテナ上のプロセスがコンテナホストの root にマッピングされている場合，ホストからコンテナプロセスに任意の情報を公開する脆弱性がコンテナランタイムに存在すると，コンテナブレイクアウトに成功する可能性が非常に高くなるとされている．Docker では userns-remap を用いることでコンテナ上のプロセスをコンテナホストの非 root ユーザにマッピングすることができる．userns-remap によって非 root ユーザにマッピングされたコンテナでは図 13 のように CVE-2019-5736 による攻撃が成功しない．

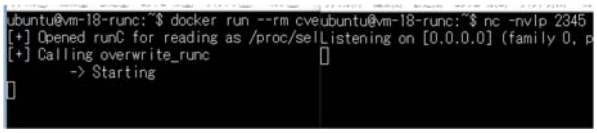

図 13　CVE-2019-5736　userns-remap の利用しコンテナ内のプロセスを非 root ユーザにマッピング（⑨）

(ii) CVE-2019-5736 では runC のバイナリを書き換える際に/usr/bin/docker-runc にアクセスしている．図 14 のように AppArmor を用いた/usr/bin/docker-runc へのアクセスの制限は CVE-2019-5736 への対応策として有効であると考えられる．このようにコンテナを用いた開発などにおいて使用し

ないと考えられるファイルやディレクトリへのアクセスは制限しておくことが望ましい．

図 14　CVE-2019-5736　AppArmor の利用によるファイルアクセス制御（⑦）

（iii）Seccomp を利用してコンテナ内で利用可能なシステムコールをフィルタリングすることは，CVE-2019-5736 への対応策として有効であるとされている．図 15 ではコンテナ内での/proc/self/exe へのシンボリックリンクの作成をフィルタリングしている．このようにコンテナ内でのシステムコールを制限しておくことはコンテナセキュリティの向上につながると考えられる．

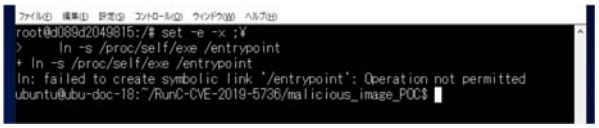

図 15　Seccomp の利用によるシステムコールの制限（⑩）

6.6 特権コンテナの悪用

Docker の Capability では privileged という最も高い権限をコンテナに付与することができ，制限されているシステムコールの利用などが可能となる．privileged の権限が付与されたコンテナを特権コンテナと呼び，本節では特権コンテナの性質を悪用し，コンテナホストの root 権限でコマンドを実行する．攻撃コードの実行には metaploit に含まれる docker_privileged_container_escape.rb を用いた．

図 16 では”whoami>/root/exploited_completed”というコマンドを実行した．図 16 右のコンソールでは/root ディレクトに exploit_completed というファイルが作成者：root として作成されており，ファイルの中身が”root”であることから，非特権ユーザが特権ユーザの権限でコマンドを実行できていることがわかる．

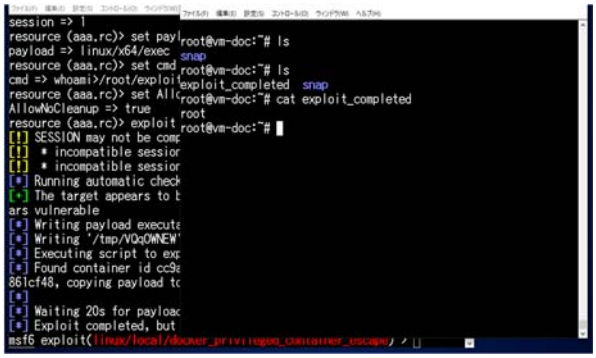

図 16　特権コンテナ（privileged）の悪用（⑥）

このように Capability の設定を誤ると，悪用される可能性があり，不用意にコンテナに対して高い権限を与えることは危険であるため，必要最低限の権限のみを付与することが望ましい．

7. 制限事項

本論文の環境において Hardware-Based の演習を行う場合，ハイパーバイザのゲスト OS 上のコンテナからのハードウェアへ正常にアクセスできるのか調査したうえで運用する必要がある．

OpenStack の脆弱性に関する演習を行う場合，OpenStack のゲスト OS に OpenStack をインストールすることによって演習可能であると考えられるが，演習内容によってはシステム全体に影響が及ぶ可能性があるため注意が必要である．

Wi-Fi など外部の物理的な環境に依存するような演習については本論文の手法では演習できない．

また，コンテナ型仮想化のみを用いた演習環境と提案手法において，同等の環境を構築し比較した場合，本論文の手法の方がより多くのリソースを消費する．

8. まとめ

（I）コンテナ内のアプリケーションからコンテナを保護する，（II）コンテナ間の保護，（III）コンテナからホストを保護する，（IV）悪意のあるまたは不誠実なホストからコンテナを保護する，といった文献[9]のコンテナセキュリティの分類において，（I）～（III）のソフトウェアベースの対策を理解する演習を網羅的に行うことを可能とするために，本論文ではコンテナとハイパーバイザを併用する演習システムの構築を提案した．そして，提案システムで攻撃手法とその対策を実装することで，（I）～（III）に含まれる 10 種類の対応策すべてを学習するコンテンツの組み合わせが存在することを示し，ソフトウェアベースのコンテナの保護手法を網羅的に学習する際に演習システム自体に支障が生じないことを確認した．

なお，ハードウェアベースの対策に分類される演習に関しては，ハイパーバイザのゲスト OS 上のコンテナからのハードウェアへのアクセスがコンテナ型仮想化のみの場合と同等とみなせるかどうかを調査する必要があるため今後の課題とする．

謝辞　本研究は総務省の「電波資源拡大のための研究開発(JPJ000254)」における委託研究「電波の有効利用のための IoT マルウェア無害化／無機能化技術等に関する研究開発」によって実施した成果を含みます．

参考文献

[1] 寺嶋友哉, 小出洋: “分散環境における拡張性を持つサイバーレンジの構築手法の提案と評価”, 情報処理学会火の国シンポジウム 2020 予稿集 (2020)

[2] Arvind S. Raj and Bithin Alangot and Seshagiri Prabhu and Krishnashree Achutha: "Scalable and Lightweight CTF Infrastructures Using Application Containers" Prerecorded Presentation,2016 USENIX Workshop on Advances in Security Education (ASE16) (2016)

[3] “2021 年 国内コンテナ／Kubernetes に関するユーザー導入調査結果を発表”, IT 専門調査会社 IDC Japan 株式会社, 2021/4/15 発行, 閲覧日：2022/5/19, https://www.idc.com/getdoc.jsp?containerId=prJPJ47597721

[4] “Threat Alert: Kinsing Malware Attacks Targeting Container Environments”, Aqua Security Software Ltd., 2020/3/3 発行, 閲覧日：2022/5/19, https://blog.aquasec.com/threat-alert-kinsing-malware-container-vulnerability

[5] Razvan Beuran, Cuong Pham, Dat Tang, Ken-ichi Chinen, Yasuo Tan, Yoichi Shinoda: “Cybersecurity Education and Training Support System: CyRIS”, IEICE Transactions on Information and Systems, E101.D(3): 740-749, 2018.

[6] 村木優太, 上原哲太郎: “サイバーレンジ演習環境展開の高速化手法”, 情報処理学会研究報告, Vol.2018-CSEC-83, No.1, pp.1-7（2018）

[7] 寺嶋友哉, 仲山昌宏, 横山輝明, 小出洋: “KAKOI: クラウドを利用したサイバーレンジをシンプルかつ安全に構築する新しいツール”, 情報処理学研究報告, Vol.2021-CSEC-93 No.15, pp.1-9（2021）

[8] 中田亮太郎, 大塚玲: “情報セキュリティ演習環境サイバーレンジへのコンテナ型仮想化活用の提案”, 情報教育シンポジウム論文集, pp.198-205（2019）

[9] Sari Sultan, Imtiaz Ahmad, Tassos Dimitriou: “Container Security: Issues, Challenges, and the Road Ahead”, *IEEE Access*, vol.7, pp.52976-52996 (2019)

[10] Ryotaro Nakata, Akira Otsuka: “CyExec*: A High-Performance Container-Based Cyber Range With Scenario Randomization”, *IEEE Access*, vol.9, pp.109095-109114 (2021)

[11] “OpenStack: OpenStack Source Cloud Computing Infrastructure”, OpenStack Infrastructure, 閲覧日：2022/5/19, https://www.openstack.org/

[12] Nestoras Chouliaras, George Kittes, Ioanna Kantzavelou, Leandros Maglaras, Grammati Pantziou, Mohamed A. Ferrag, “Cyber Ranges and TestBeds for Education, Training, and Research”. *Applied Sciences*, vol.11, no.4 (2021)

[13] “Docker: Home”, Docker Inc., 閲覧日：2022/5/19, https://www.docker.com/

[14] “Oracle VM VirtualBox”, Oracle Corporation and/or its affiliates, 閲覧日：2022/5/19, https://www.virtualbox.org/