Red Hat Enterprise Linux Atomic Host 7

コンテナーセキュリティーガイド

コンテナーセキュリティーガイド

Last Updated: 2018-02-07

Red Hat Enterprise Linux Atomic Host 7 コンテナーセキュリティーガイド

コンテナーセキュリティーガイド

法律上の通知

概要

コンテナーのセキュリティーに関するガイド

目次

パート I. コンテナーセキュリティーガイド

第1章 概要

本書では、Docker ワークフローをよりセキュアにする方法について説明します。ワークフローのセキュリティー保護については、すべての問題に対応するような単一ソリューションはありません。Docker ワークフローのセキュリティー保護に関連する問題を一般化できるアルゴリズムに落とすことができるなら、Red Hat はそのアルゴリズムをソフトウェアに組み込み、RPM としてパッケージ化し、これを RHEL に組み込んでからそのサポートを販売することでしょう。

しかしながら、Docker ワークフローのセキュリティーを保護するための唯一の万能薬のようなソリューションはないため、Docker ワークフローのセキュリティー保護を強化するために使用できるツールおよび戦略を理解しておく必要があります。これを実行するための一般的な戦略として、悪意あるエージェントが利用できる Docker インフラストラクチャー内の潜在的な攻撃ベクトルを減らすことができます。

本書では、セキュリティー対策と Docker エコシステムとの関係を理解するために必要な基本情報を記載します。本書を熟読していただくと、お使いの Docker ワークフローに最適なセキュリティーソリューションを見つけるために必要なツールおよび戦略情報を得ることができます。

各種のセキュリティー対策に通じているユーザーであれば、本書の目次をご覧になるだけで、コンテナーのセキュリティー保護の一般的な戦略をご理解いただけることでしょう。基本的には、信頼されていないコンテンツを避けること、コンテナーを最新の状態に保つこと、SELinux を活用すること、攻撃対象領域を最小限に抑えること、および仮想化を使用して「アパートメント (apartment)」を作成することなどが必要になります。

コンテナー化のエコシステムは開発の途にあり、急速な変化を遂げています。本書は、お客様がコンテナー化の独自の実装において最適なオプションを選択できるようにコンテナー化のエコシステムを理解しやすく説明することを目的としています。

第2章 コンテナーの構築およびセキュリティー保護

本章では、Docker コンテナーの構築および配布に関するセキュリティー上の懸念点について記載します。

Docker の未署名のバイナリー

警告

Docker の公式のバイナリーインストールは署名されていません。この点については本書が公開される時点で進展があることが期待されますが、変更がない場合には Docker の公式インストールは署名されていないという点に留意してください。

信頼されないコンテンツの危険性

RPM を使用するプロセスは、インストールの取得フェーズとインストールの実際のインストールフェーズに分かれています。しかし、Docker ワークフローでは取得とインストールが分かれていません。この問題に関連する CVE は多数あります。Docker イメージは (ほとんどの場合) tar として保存され、それらのイメージは ユーザーが認識しないところで Docker デーモンをエスケープする場合があります。

- Docker のプルはアクティブなプロセスです。RPM とは異なり、プロセスは分離されません。
- Docker コンテナーは root で実行されます。信頼されているベンダー (Red Hat はその代表例) からのみ提供される Docker コンテンツを実行する必要があります。

Docker Hub のコンテナーは管理対象外

http://www.banyanops.com/blog/analyzing-docker-hub/ : コンテナーは Docker Hub にアップロードされますが、アップロード後のコンテンツはメンテナンスは Docker Hub では実行されません。メンテナンスは、Docker デーモンホストに参加するイメージ作成者の勤勉さとインテリジェンスにもっぱら依存することになります。

第3章 コンテナーを最新かつ更新可能な状態にする

このセクションでは、コンテナーを最新で更新可能の状態にするプロセスと方法について説明します。

3.1. 素の PULL を FROM 命令に入れない

使用する FROM コマンドにプルするレジストリーを常に一覧表示します。つまり、Red Hat の場合は Red Hat Docker レジストリーの名前全体を組み込む必要があります。

以下は素の pull です。

```
$ docker pull rhel7
```

以下は素の pull ではありません。

```
$ docker pull registy.redhat.com/rhel7
```

3.2. YUM UPDATE ツールの使用

ベースイメージが最新になるように **yum update && yum clean all** または同等のコマンドを常に実行します。これらのコマンドは、ベースイメージに関連付けられた RPM コンテンツを更新します。これにより、攻撃対象となる可能性のある領域が縮小します。

3.3. DOCKER キャッシュの活用

このセクションでは、Dockerfile をワークフローに合わせて効率化するために Docker キャッシュを使用する方法について説明します。

3.3.1. キャッシュを利用するための順序命令

Docker はそれぞれの命令が確定的であると仮定します。さらに Docker はこれらの命令には関連性がないと仮定します。Docker は同じ命令が同じ順序で出されると、結果をキャッシュします。つまり FROM foo: dnf -y update の命令が 2 つの同じ Dockerfile に同じ順序で存在する場合、Docker はその命令が出される時点から同じベースイメージを作成します。

Docker キャッシュを活用するには、Dockerfile によるインストールを常に同じ順序で実行してください。Docker のキャッシュユーティリティーを最大限に活用するには、タスクをテーマごとのコンポーネント (例: 「ユーザーの追加」または「ソフトウェアのアップグレード」) に分割してください。時間とリソースに余裕がある場合は Dockerfile スタイルガイドを作成することをお勧めします。

3.3.2. キャッシュを意図的に分割する必要のある状況

Docker のデフォルトのキャッシュ動作を無効にする必要がある状況があります。このセクションではどんな場合に何を実行する必要があるのかについて説明します。

次のシナリオを考えてみましょう。アップデートが必要であると認識しているものの、Docker は "yum update" が同じ状態を返すと誤って仮定しているため、そのアップデートは Dockerfile のアップデートでは扱われません。しかし、"yum update" は確定的なコマンドではないため、いつも同じ状態を返す訳ではありません。以下は、アップデートを強制実行する 3 つのメソッドです。(1) 単一マシンの場合は、イメージを削除し、再作成する。(2) キャッシュを意図的に分割する "echo uniquething" などのナ

ンス (nonce) コマンドを挿入する。(3) OpenShift ビルドシステムのようなビルドシステムを使用する。このビルドシステムはこの問題を認識して OSBS (OpenShift ビルドシステム) から新規イメージを要求することを許可します。

3.4. コンテナーの出所の確認

コンテナーの出所を確認するためにできる最も簡単なことは、プライベートレジストリーを各自で実行することです。OpenShift および Satellite はどちらも、組み込みイメージサービスオプションを持つ Red Hat 製品です。お使いのイメージには機密情報を含めないことをお勧めします。機密情報がなければ、誰かがイメージをたまたま閲覧することがあっても問題にはなりません。つまり、TLS (トランスポート層セキュリティー) を常に使用する必要がありますが、認証を使用する必要はありません。機密情報として保持する必要のある情報はオーケストレーションレベルまで抽象化される必要があります。これについては、本書の Kubernetes および OpenShift のセクションで論じます。

3.4.1. レジストリーなしの Docker イメージの配布

レジストリーなしに Docker イメージを配布するための最も良い方法は何でしょうか。Docker イメージは "docker save" コマンドで作成され、"docker load" でロードされる tarball として配布できます。これらの出所のセキュリティーは、他のセキュリティー保護の場合と同様に署名付きハッシュを使用して保護できます。

http://fedora.uberglobalmirror.com/fedora/linux//releases/21/Docker/x86_64/: fedora の場合、このリンクに示されるようにハッシュファイルとそのハッシュファイルに関連付けられた tarball があります。

3.5. コンテナーを不変にするために KUBERNETES および OPENSHIFT を活用する

不変性がセキュリティーにとって重要なのはなぜでしょうか。不変性により、ローカル侵害によって加えられる可能性のある損害が軽減されます。イメージ自体には秘密情報が含まれず、破損する可能性のある状態は保存されません。さらにそれらのイメージは変更されないため、確認が必要な箇所はわずかになります。

(この文脈での「不変 (immutable)」とは、変更される状態を持たないコンテナーを意味します。)

3.5.1. Kubernetes および OpenShift を活用する方法

このセクションでは、不変な (ステートレスな) コンテナーイメージを作成するために Kubernetes および OpenShift を活用する方法について説明します。

1. ボリュームマウントの使用: 変更可能な外部データ (WordPress コンテンツまたはデータベースなど) を取り込みます。

2. サービスの使用: Kubernetes および OpenShift にはマッチメイキング (matchmaking) サービスがあります。コンテナーは、一般的にデータベースに依存させるように設計でき、その場合データベースへのログインの詳細情報はランタイム時に提供されます。

3. テンプレートの作成: これは上記のアイデアと同じですが、ビルド時に適用されます。コンテナーが特定のクラスターを実行するためにユーザーに特定の UID を持たせる必要がある場合、Dockerfile ではこれを実行することができません。このタスクを実行する必要がある場合は、OpenShift ビルドシステムへのプラグインを使用して所定のビルドをカスタマイズすることができます。

4. Github リポジトリーの使用: docker pull を使用して、ランタイム時にプライベートまたはパブリック git リポジトリーのライブコンテンツをプルします。OpenShift には、これをさらに別の

レベルへと引き上げる機能があります。これらの機能により、すべてのコンテナー詳細情報を無視し、すべての詳細情報が github リポジトリーでホストされるアプリケーションを使用できます。

3.5.2. コンテナーが秘密情報または機密情報を保存しないように Kubernetesを活用する

Kubernetes には、メモリーでホストされる秘密情報をランタイム時に仮想ファイルとして挿入する「シークレット (secrets)」機能があります。これは、認証の詳細情報や暗号化キーなどのすべての秘密情報に使用されます。

第4章 コンテナー分析ツール

このセクションでは、コンテナーの分析に使用するツールについて説明します。

4.1. OPENSCAP

これは仮想マシンの評判の高い openSCAP プロジェクトの拡張機能です。OpenSCAP は、コンテンツで既知の CVE、古くなったコンテンツおよび不正な権限などの様々なコンテンツ関連の問題を検索します。これはコンテナーイメージに対するセキュリティー対策のサニティーチェックです。

4.2. ATOMIC コマンド

「atomic」コマンドを使用すると、イメージに関連する層を認識でき、それらの層のいずれかが更新される場合にイメージの再構築が必要であることを確認できます。

第5章 コンテナーでロックダウンされたセキュアな FIREFOX

このセクションでは、Firefox を実行するセキュアなコンテナーをデプロイする方法について説明します。このコンテナーは、以下の機能を含むコンテナー化された Firefox のインスタンスを提供します。

- 特権は完全に不要: SELinux の追加の調整は不要です。
- cgroup の一覧のみがホストからコンテナーに渡されます。
- コンテナーはホストのみで利用できるため、ポートはリダイレクトされません。
- X11 クリップボードイベントまたは X イベントは実際のホストと共有されません。
- 共有されるサウンドハードウェアはありません。
- systemd 以外のすべてが通常の特権のないユーザー権限で実行されます (また systemd は他のプロセスの reap (シャットダウン) を実行するためにのみ実行されます)。
- (同期なし) サウンド、flash、および良好な対話機能。

コンテナーでの Firefox の安全な実行

ステップ 1

コンテナーを構築するために使用するベースイメージを取得します。

```
$ curl -o Fedora-Docker-Base-22-20150521.x86_64.tar.xz -L
https://download.fedoraproject.org/pub/fedora/linux/releases/22/Docker/x
86_64/Fedora-Docker-Base-22-20150521.x86_64.tar.xz
```

ステップ 2

ダウンロードしたばかりのベースイメージをローカルの Docker レジストリーにロードします。

```
$ sudo docker load < Fedora-Docker-Base-22-20150521.x86_64.tar.xz
```

ステップ 3

このコンテナーをマップする Dockerfile を保持するためのディレクトリーを作成します。

```
$ mkdir -p isolated_firefox
```

ステップ 4

以下の curl コマンドを使用して Dockerfile を取得します。

```
$ curl -o isolated_firefox/Dockerfile -L http://pastebin.com/raw.php?
i=cgYXQvJu
```

ステップ 5

コンテナーを構築し、**`isolated_firefox`** というタグでこれにタグ付けします。

```
$ sudo docker build -t isolated_firefox isolated_firefox
```

ステップ 6

コンテナーを実行します。

```
$ sudo docker run -v /sys/fs/cgroup:/sys/fs/cgroup:ro isolated_firefox
```

ステップ 7

docker ps コマンドを使用して CONTAINER_ID を取得します。

```
$ sudo docker ps
```

ステップ 8

コンテナーの IP アドレスを取得します。

```
$ sudo docker inspect CONTAINER_ID| grep IPAddress\":
```

ステップ 9

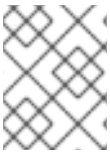
vncviewer でコンテナーを開きます。

```
$ vncviewer CONTAINER_IP
```

ステップ 10

このコンテナーに関連付けられているオーディオを聞くには、ブラウザーを開いて以下の場所に移動します。

```
http://CONTAINER_IP:8000/firefox.ogg
```



ポートを URL に組み込むことを忘れないでください。

つまり、URL の後に **:8000** を入力し忘れないでください。さらに VLC でコンテンツを再生するためにコンテナーのアドレスを VLC に送信することもできます。

VLC インスタンスを起動するために以下のコマンドを実行します。

```
$ vlc http://CONTAINER_IP:8000/firefox.ogg
```

第6章 DOCKER SELINUX セキュリティーポリシー

Docker SELinux セキュリティーポリシーは libvirt セキュリティーポリシーに類似しており、libvirt セキュリティーポリシーに基づいています。

libvirt セキュリティーポリシーは、仮想マシンを分離するための 2 つの方法を定義した一連の SELinux ポリシーです。通常、仮想マシンはネットワークの特定の部分にアクセスできないようにされます。具体的には、個々の仮想マシンは相互のリソースへのアクセスが拒否されます。Red Hat は libvirt-SELinux モデルを Docker に拡張しています。Docker の SELinux ロールおよび Docker の SELinux タイプは libvirt に基づいています。たとえば、デフォルトで Docker は /usr/var/ およびその他の場所にアクセスできますが、svirt_sandbox_file_t でラベル付けされたものには完全なアクセスがあります。

https://www.mankier.com/8/docker_selinux: Docker の SELinux ポリシー全体について説明しています。これは専門用語を使って説明されており、詳細情報が記載されています。

svirt_sandbox_file_t

```
system_u:system_r:svirt_lxc_net_t:s0:c186,c641
```

```
^      ^           ^          ^     ^--- unique category
|      |           |          |----  secret-level 0
|      |           |--- a shared type
|      |---SELinux role
|------ SELinux user
```

ファイルに "svirt_sandbox_file_t" のラベルが付けられると、デフォルトですべてのコンテナーがこのファイルを読み取ることができます。ただし、これらのコンテナーが "svirt_sandbox_file_t" 所有権を持つディレクトリーに書き込む場合、独自のカテゴリー (この場合は "c186,c641) を使用して書き込みを実行します。同じコンテナーを 2 回起動すると、2 回目には新たなカテゴリー (初回のカテゴリーとは異なるカテゴリー) が取得されます。このカテゴリーシステムにより、コンテナーは相互に分離されます。

各種のタイプはプロセスおよびファイルに適用できます。

6.1. LIBVIRT SELINUX

libvirt SELinux: RH は Docker 用に libvirt SELinux を拡張しています。libvirt SELinux の Docker への拡張は、そのほぼすべてが RH によって提供された機能です。

6.2. MCS (マルチカテゴリーセキュリティー)

MCS (マルチカテゴリーセキュリティー): これはマルチレベル認証に似ています。各コンテナーには起動時に固有 ID が付与され、コンテナーが書き込む各ファイルにはその固有 ID が継承されます。コンテナーのファイルは、いわばコンテナーの DNA から逃れることができません。これはオプトインシステムですが、その使用は得策と言えます。この機能を使用しないと、コンテナー間を分離することができません。コンテナーとホスト間の分離は、この機能の使用にかかわらず可能ですが、コンテナーを相互に分離することはできません。つまり、(MCS を使用しない場合は) あるコンテナーが別のコンテナーのファイルにアクセスできる可能性があります。

https://securityblog.redhat.com/2015/04/29/container-security-just-the-good-parts/: この情報は、後に MCS に組み込む MCS サンプルを構築する際に使用します。

6.3. DOCKER SELINUX セキュリティーモデルの活用

コンテンツの適切なラベル付け: デフォルトでは Docker は /usr のすべて、および /etc のほとんどにアクセスできます。これ以上のアクセスを付与する必要がある場合は、ホスト上のコンテンツを再度ラベル付けする必要があります。/usr や /etc にあるものへのアクセスを制限する必要がある場合も、アクセスを制限するコンテンツを再度ラベル付けする必要があります。なお 1 つまたは 2 つのコンテナーのみにアクセスを制限する場合は、オプトインの mcs システムを使用する必要があります。

重要なブール値およびその他の制限: Docker における「特権」は実際の特権を意味するものではありません。特権付きの Docker プロセスも任意のソケットファイルにアクセスできません。selinux ブール値の docker_connect_any を使用すると、特権付きの Docker プロセスは任意のソケットファイルにアクセスできます。Docker は特権付きで実行されている場合でも、有効なブール値によって制限されます。

docker_connect_any: 上記の『重要なブール値およびその他の制限』を参照してください。

https://github.com/rhatdan/docker-selinux: dwalsh の github リポジトリーです。この機能はユーザーが利用できるように man ページに記載する必要があります。

カーネル機能の制限: Docker は "docker run" の一部として (1) "--cap-add=" と (2) "--cap-drop=" の 2 つのコマンドをサポートします。これらにより、カーネル機能をコンテナーに追加したり、コンテナーからドロップしたりできます。root 権限は数多くの機能グループに分割されます (ファイルの所有権を変更する "cap-chown" など)。デフォルトの Docker 機能一覧は非常に限られています。

http://linuxmanpages.net/manpages/fedora21/man7/capabilities.7.html: この情報は機能の詳細情報を必要とするユーザー向けです。これらの機能はすべてコンテナーの分離に関連するものです。これらの機能を適切な方法 (本書で説明されている方法) で使用するなら、システムにカーネルの脆弱性がない限り、攻撃者はシステム上で root アクセスを利用することはできません。すべての機能をドロップする場合、root は権限がない状態となり、機能をオフにして無効にした root アカウントを所有する攻撃者は何の影響も与えることはできず、保護対象外のものしか操作できません。

seccomp によるカーネル呼び出しの制限

http://man7.org/linux/man-pages/man2/syscalls.2.html: これは期待を起こさせる機能です。seccomp は各種の機能よりも詳細化されています。この機能を使用すると、コンテナーが実行する実際のカーネル呼び出しを制限することができます。これは、(たとえば) コンテナーが "cd" を呼び出すことを防ぐことができるため、一般的なセキュリティー上の理由からも期待できる機能であると言えます。ほとんどすべてのカーネルの不正利用は、カーネル呼び出しに伴って生じます (カーネルのほとんど使用されていない部分に対して実行されるのが通例)。seccomp を使用すると数多くのカーネル呼び出しをドロップでき、ドロップされたカーネル呼び出しは攻撃ベクトルとして悪用できません。

Docker ネットワークセキュリティーおよびルーティング: デフォルトで、Docker は各コンテナーの仮想イーサネットカードを作成します。各コンテナーには独自のルーティングテーブルと iptables があります。このほかにも、特定ポートの転送を要求する場合に Docker は特定のホストの iptables ルールを作成します。Docker デーモンは自らプロキシー機能の一部を実行します。アプリケーションをコンテナーにマップすると、アプリケーションごとにネットワークアクセスを制限できるため、柔軟性を確保することができます。

シナリオ: 特定の種類のインターネットトラフィックを他のインターネットトラフィックから分離するにはコンテナー化されたファイアウォールを使用します。読者の演習例として、特定の種類のトラフィックを正式なネットワークから分離するシナリオを想定することができるかもしれません (配偶者または雇用者の監視下に置かれるトラフィックなど)。

cgroup: これはコントロールグループのことです。これらは Docker の実行内容において中心的な役割を担っています。元々 cgroup は cpu などのリソースに対してのみアクセスを制御するものでした。プロセスを cgroup に入れてからカーネルに対して「その cgroup には cpu の最大 10 パーセントを割り当てる」ように指示できます。これは一種の SLA またはクォータの提供方法です。デフォルトで、

Docker は各コンテナーに固有の cgroup を作成します。Docker デーモンホストに既存の cgroup ポリシーがある場合は、その既存の cgroup ポリシーを使用して指定コンテナーのリソース消費を制御できます。

コンテナーのフリーズおよびフリーズ解除: ある時点の状態にあるコンテナーをフリーズし、その時点からコンテナーを再起動することができます (コンテナーの cpu を 0% に指定すると効果的に実行できます)。cgroup は Docker が DDoS 攻撃に対して提供する保護機能になります。マシン上でサービスをホストし、このサービスに cgroup の優先順位を指定できます。これにより、サービスの cpu パーセントが10% を割らないようにし、かつ他のサービスがセキュリティー侵害の危険にさらされても、それらのサービスが最低 10% の cpu レベルを維持するこのサービスに影響を与えないようにします。そのため、必須のプロセスがどのような悪意ある方法で攻撃されたとしても、それらのプロセスが cpu の一部を制御できなくなる状況を避けることができます。/sys/fs/cgroup は仮想ファイルシステムです。デフォルトでは、コンテナーは /sys/fs/cgroup にアクセスできません。

第7章 コンテナーのセキュリティー対策

本章では、コンテナーのセキュリティーを保護するために実施できる対策について説明します。

7.1. カーネル機能のドロップ

カーネル機能は Docker の CLI から、および Kubernetes の .json ファイルを使ってドロップできます。ゼロの状態から必要な機能を追加できます。これは、必要なカーネル機能を判別するための最も安全な方法です。

http://blog.siphos.be/2013/05/capabilities-a-short-intro/: Sven Vermeulen 氏によるこのブログ掲載では、各種機能の基本について説明しています。Vermeulen 氏は、anotherping というファイルに ping バイナリーをコピーした後は ping バイナリーには生パケットの送信や、CAP_NET_RAW 機能 (生パケットの送信を許可) の割り当てを許可する機能がなくなり、CAP_NET_RAW 機能が割り当てられた anotherping で生パケットを送信できるようになることを説明しています。

手順 4.1. 機能の実証

```
1.

#         cp ping anotherping

2.

#         chcon -t ping_exec_t anotherping

3.

$         ping -c 1 127.0.0.1     NG 127.0.0.1 (127.0.0.1) 56(84) bytes
of data.
 64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.057 ms

4.

$         anotherping -c 1 127.0.0.1    ping: icmp open socket: Operation
not permitted

5.

#         setcap cap_net_raw+ep anotherping

6.

#         anotherping -c 1 127.0.0.1    PING 127.0.0.1 (127.0.0.1) 56(84)
bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.054 ms
```

7.2. ROOT のドロップ

systemd の場合を除き、長時間実行されるコンテナーを root で実行された状態にしないでください。次のような 2 つのアプローチを実行できます。systemd が不要な場合は exec を使用して root になり、ユーザー特権をドロップできます。または systemd を使用する場合は、ユニットファイルを使ってコンテナー内で必要なアプリケーションを起動でき、そのユニットファイルを使用して非 root として必要なサービスを指定できます。

7.3. 特権の確認

コンテナーがホストのハードウェアへのアクセスを必要としない限り、--privilege は使用しないでください。

7.4. コンテナー内のネットワークセキュリティーおよびルーティング

SELinux はまもなくコンテナー内で使用可能になります。

コンテナーには独自の iptables およびルーティングルールがあり、これらを使用して、デフォルトで制限されるネットワークをセットアップし、コンテナーにアクセスすることが期待されるネットワークのみがコンテナーにアクセスできるようにする必要があります。

7.5. SUID コンテンツ

スティッキービットコンテンツ (スティッキー UID コンテンツ) のことです。以下のアドバイスに留意してください。特権コンテナーなしに suid コンテンツを使用することはできますが、特権コンテナーなしに suid コンテンツを作成することは容易ではありません。suid としてマークする操作には特権が必要となるためです。

7.6. TMPFILE

tmpfile は /tmp に作成されるファイルです。本セクションでは tmpfile 攻撃に言及します。コンテナーは、コンテナー以外の tmpfile が影響を受けるのと同じ競合状態と攻撃の影響を受けます。tmpfile コンテンツの所有権が適切に制限されていることを確認してください。

第8章 外部テクノロジーの活用

本章では、仮想化および Kubernetes ネットワークセキュリティーおよびルーティングを使用してセキュリティーゾーンを構築する方法について説明します。

8.1. 仮想化によるセキュリティーゾーンの構築

Dan Walsh 氏がこのアイデアについて言及しており、このトピックについてのブログがいずれ掲載される可能性があり、ここではとくにこの点についてコメントすることは控えます。Dan Walsh 氏の語った「containers don't contain (コンテナーは (攻撃を) ブロックする訳ではない)」という言葉は読者の皆さんもご存知のことでしょう (コンテナーは VM とは異なりローカル特権昇格の攻撃から保護されている訳ではないという意味)。Kubernetes を使用すると、異なるタイプの実行中のコードを分離する「アパートメント (apartment complexes)」(一般的に使われる表現ではありませんが、Walsh 氏の言葉を引用しています) を作成できます。たとえば、特定のセキュリティーレベルのアプリケーション用の VM があるとします。法律事務所で使用する場合、法律事務所のクライアント (ここでのクライアントは「クライアントサーバー」アーキテクチャーの「クライアント」ではなく、法律事務所にサービスを依頼するエンティティーのこと) 別に VM を用意し、弁護士達はその VM 内で実行されるコンテナーで作業するかもしれません (つまり、だれかが VM に侵入したとしても、対象領域はその VM 内のデータのみとなり、他の事例に関連するデータには影響がありません)。このような場合には、Kubernetes の上部で実行される Apache のリソース管理アプリケーションである Mesos を使用できます。

8.2. KUBERNETES の認可およびセキュリティー

これは調査の必要な分野です。認可、ログインおよびクライアント証明書の検証についてはすべて調査が必要です。

8.3. KUBERNETES ネットワークセキュリティーおよびルーティング

Kubernetes についての最も大きな懸念点とは Kubernetes クラスターをコンテナー自体からいかに保護するかという点です。コンテナーが独自のネットワークアクセスを使用し、Kubernetes に対して承認されていないコンテナーを生成するように指示し、Kubernetes がその要求に従うという望ましくない状況が生じる可能性があります。今後は「多層防御 (defense in depth)」の一部として、この種の望ましくない攻撃をネットワークレベルで停止するだけでなく、パスワードとログおよび証明書を Kubernets に導入することについての調査が行われます。コンテナーが Kubernetes 機能にアクセスする必要がない場合、Kubernetes をネットワーキング関連の要求からブロックする必要があります。コンテナーについては、ライフサイクルの初期段階で Kubernetes から必要な機能を取得したコンテナーを作成し、コンテナーの独自のファイアウォールのセキュリティーを保護できるため、潜在的な攻撃ベクトルを排除し、環境の攻撃対象領域を最小にできます。