ペパボ研究所
Pepabo R&D Institute, GMO Pepabo, Inc.

suEXEC セキュリティモデルをベースにしたコンテナ起動時のバリデーションの提案

Uchio Kondo / GMO Pepabo, Inc.
2018.05.12 WSA研究会 #2

ペパボ研究所
Pepabo R&D Institute, GMO Pepabo, Inc.

シニア・プリンシパルエンジニア

近藤宇智朗 / @udzura

Uchio Kondo

技術基盤チーム

https://udzura.hatenablog.jp

目次

1.

コンテナ環境提供の際のセキュリティ的課題

大きな前提: Webサービス提供でのコンテナの利用

- 突発的アクセス増などへの対処のため、Webサービスの運用者は高速かつ柔軟なリソース制御をしたい（急なリソース追加への対応、不要な時は減らす、など）。
- コンテナはその起動の高速さなどの性質から、その目的にあっており普及が進みつつある。
  - Haconiwa 論文(*) でもこの前提について述べている
- 今回はその中でも、ホスティング環境において高速かつ柔軟なリソース制御を実現するためにコンテナを利用するにあたり、セキュリティの問題を整理する

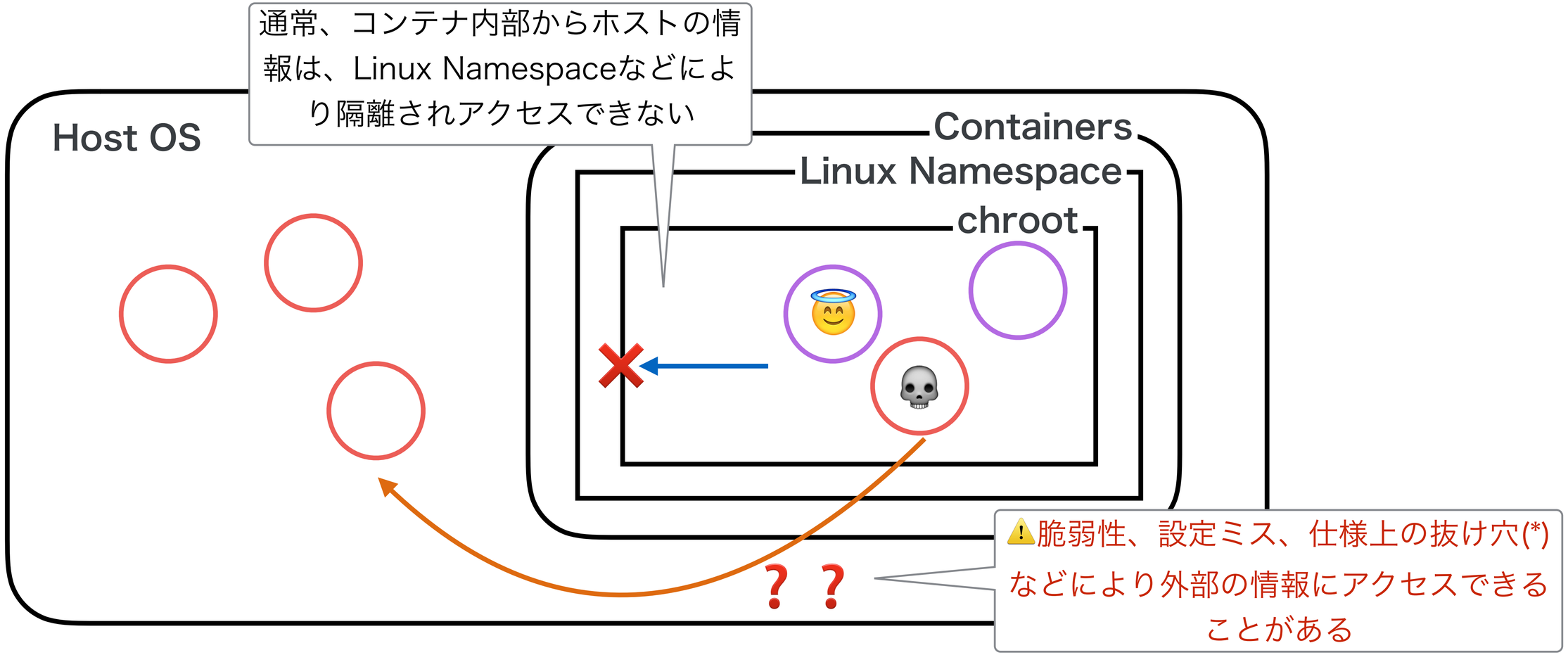
コンテナ提供時のセキュリティ的課題

1. コンテナの内部から、コンテナをホストしている親のOSに「脱獄」ができる場合
2. コンテナの内部から、コンテナの「外」が見えたり操作できる場合
3. コンテナの内部について、サービス提供者が意図しない設定変更やファイルの操作ができる場合
4. コンテナの起動時に、サービス提供者が意図しないコマンド、引数、権限での起動をされる場合

コンテナの内部→コンテナのホスト

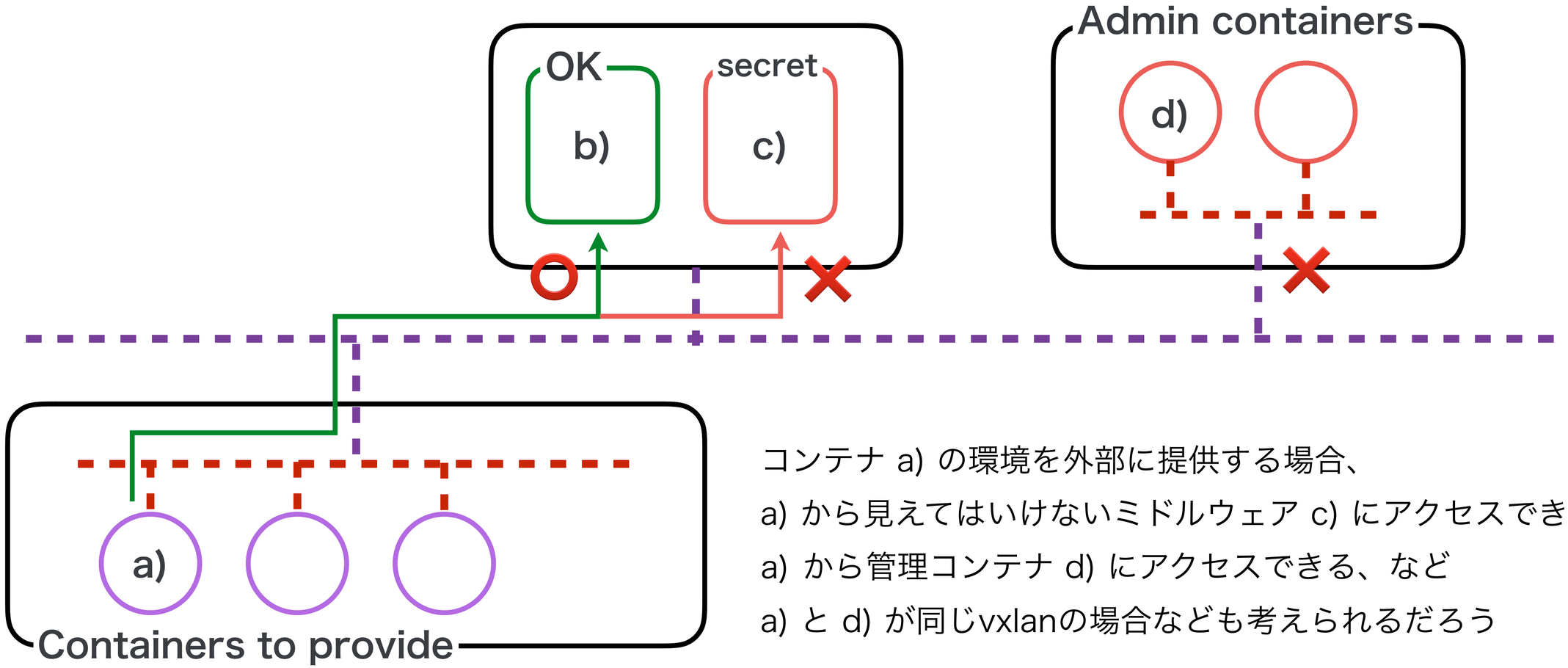
(*) unchroot: https://gist.github.com/FiloSottile/697618

- いわゆる「脱獄」



コンテナの内部→見せたくない外部

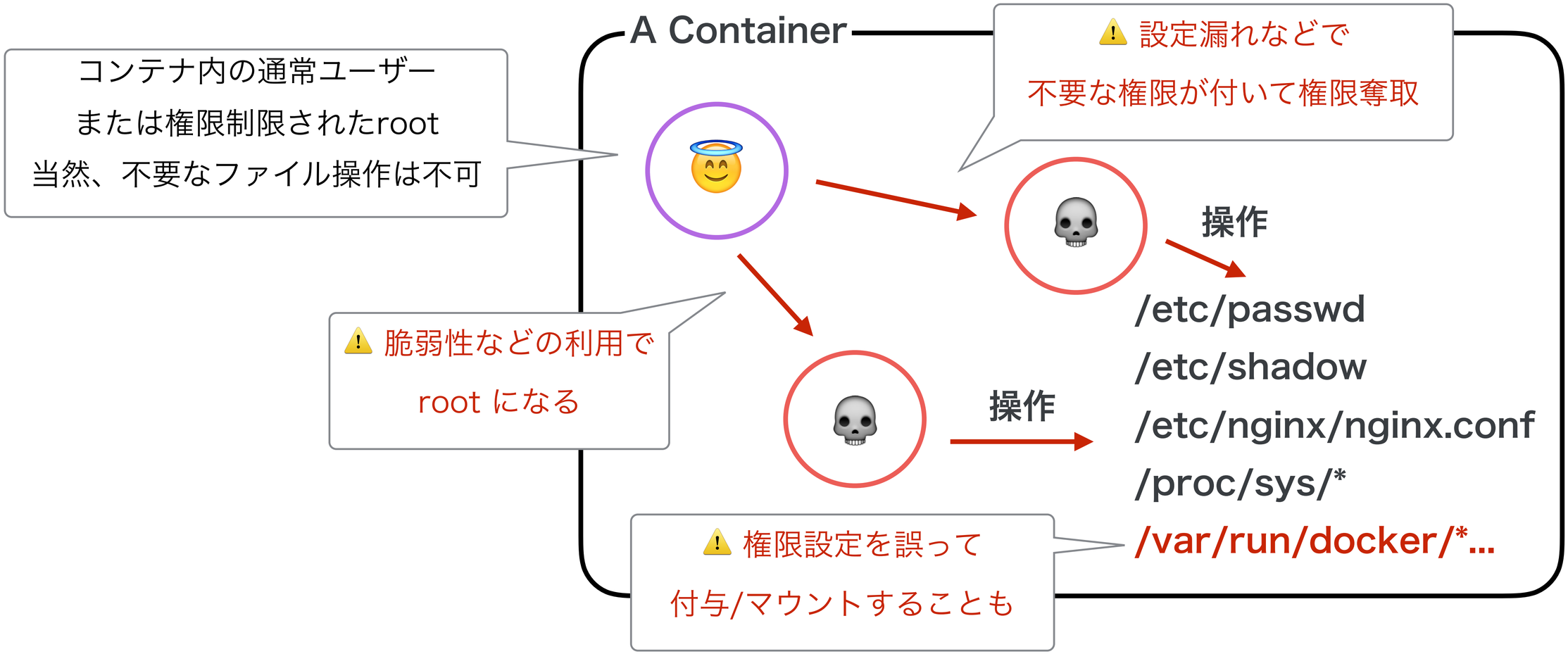
- ネットワーク上の他のホストへのアクセスなど



コンテナ a) の環境を外部に提供する場合、
a) から見えてはいけないミドルウェア c) にアクセスできる、
a) から管理コンテナ d) にアクセスできる、など
a) と d) が同じvxlanの場合なども考えられるだろう

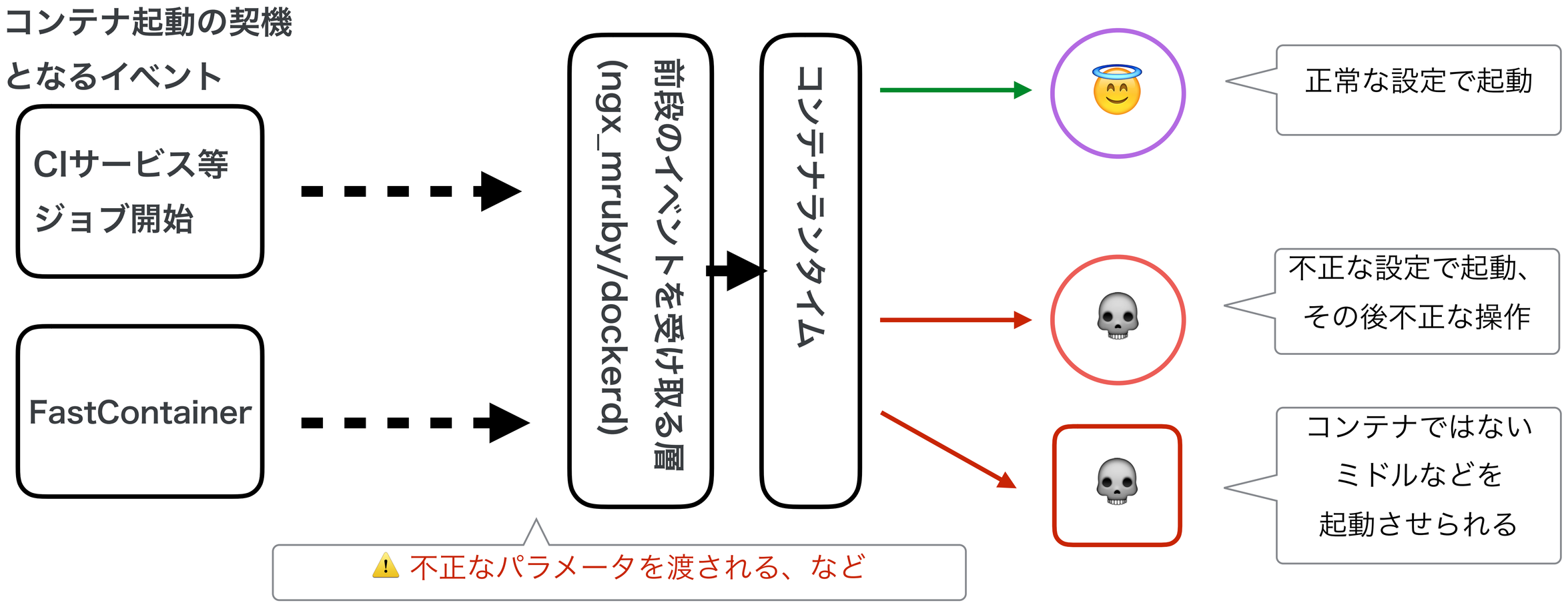
コンテナの内部を越権的に操作できる

- コンテナ内部の特権を与えていないのに、ファイルを操作できる



意図しないオプション・権限でコンテナが作られる

- コンテナの起動をユーザでコントロールできる場合がある



2.

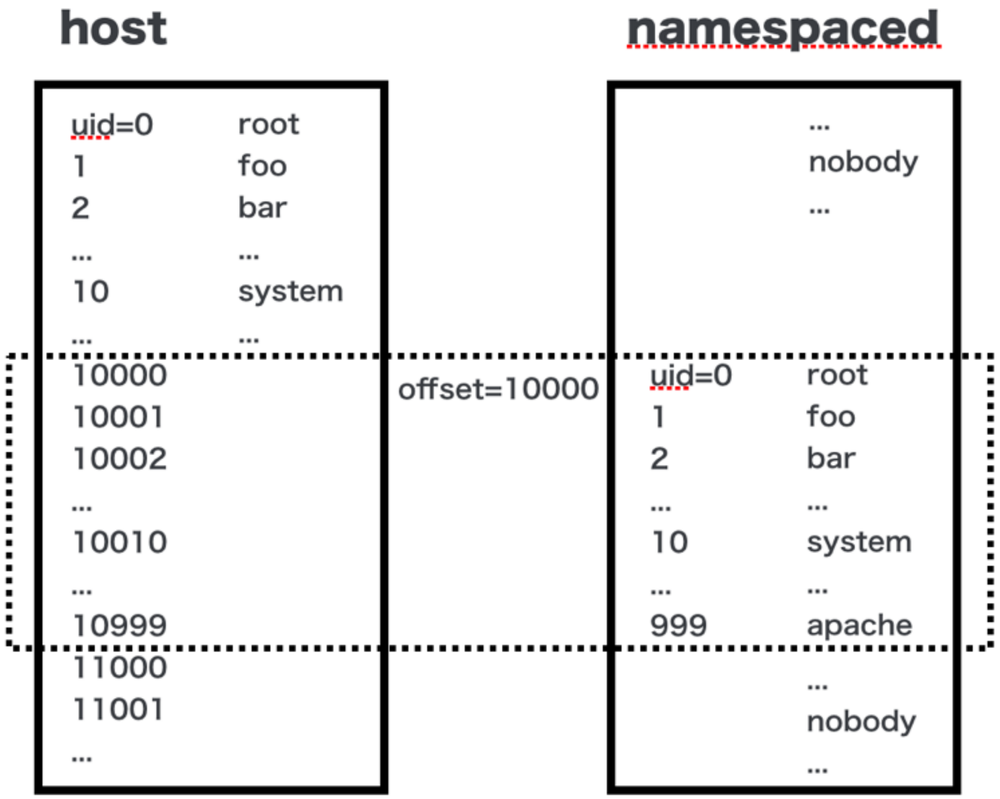
コンテナにまつわる
セキュリティ機構の整理

コンテナに関するセキュリティ機構を一覧する

1. User Namespace
2. Kernel Capabilities
3. seccomp
4. Mandatory Access Control
5. Service Meshing
6. Rootless/Unprivileged containers
7. (Hypervisors)

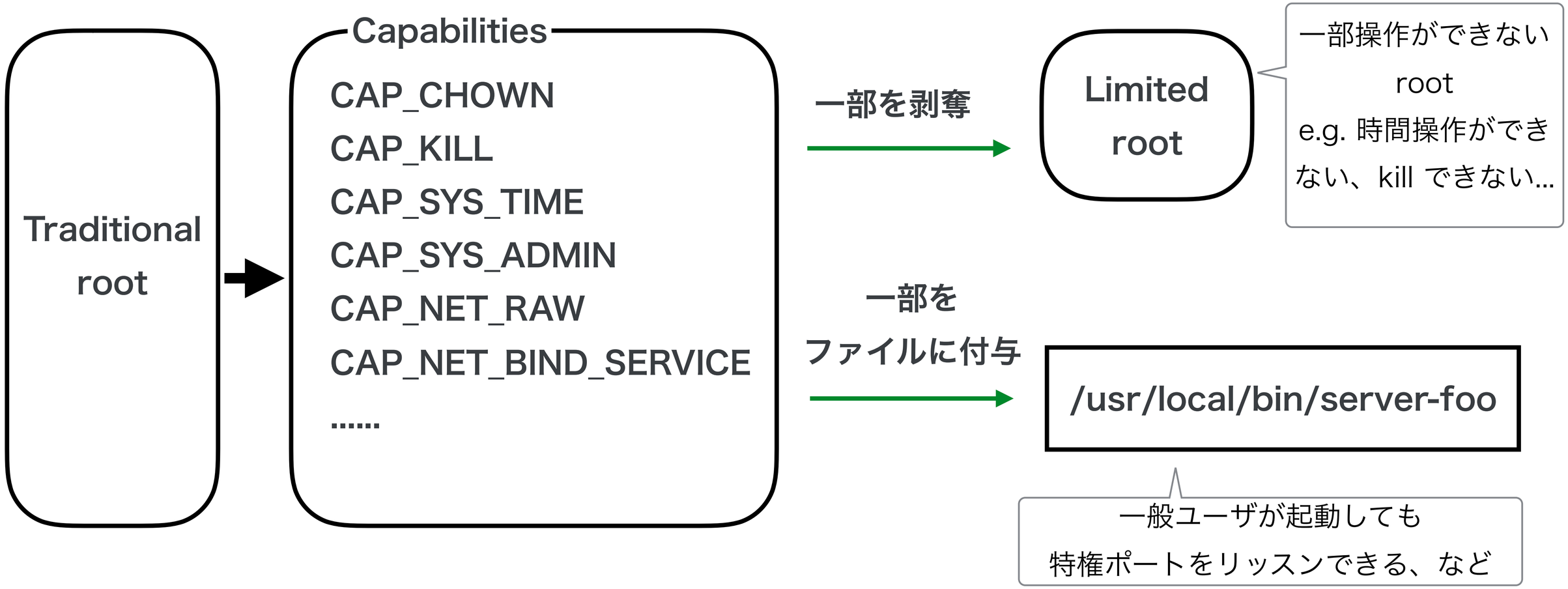
User Namespace

- ホスト名などのように本来OSグローバルに存在するリソースについて、
  OS内部で名前空間を作成し、
  複数を存在させ使い分ける
  (Linux Namespace)
- ユーザIDについて名前空間を
  作り、ホストの非rootを
  コンテナの管理者に。



Kernel Capabilities

- rootの持つ特権を、分割し、一部を剥奪あるいは付与できる

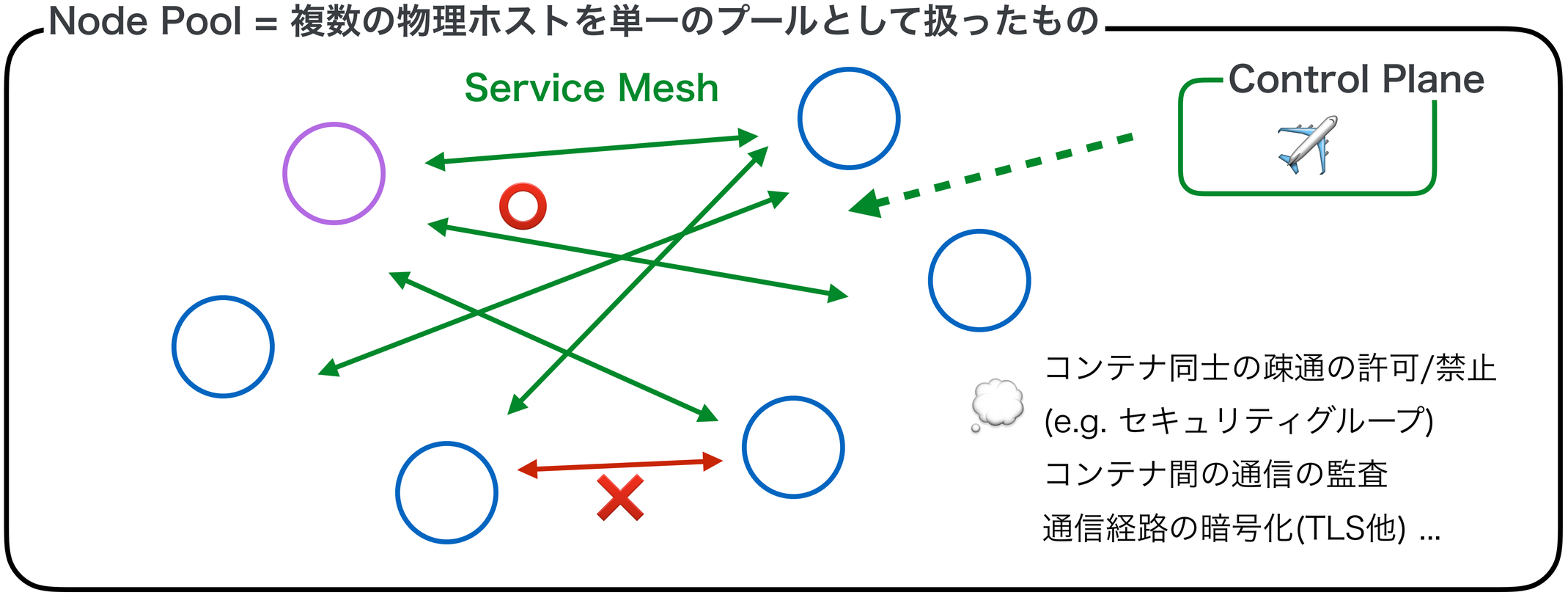


seccomp / Mandatory Access Control

- ユーザの今の権限にかかわらず、一部操作を制限・監査する
- seccompであればシステムコール、MACであればファイルアクセスなど

Service Meshing

- コンテナ同士の相互のネットワーク、疎通を、コードなどによりコントロールすることをService Meshと呼び、コンテナの文脈でよく使われる



Rootless/Unprivileged containers

- 1) バイナリ自体を set-user-id root する
- 2) LXC 方式の非特権コンテナ（User Namespaceを利用）
- 3) 須田氏によるRootless runcほか（機能制限あり）
- メリット・デメリットまとめ:

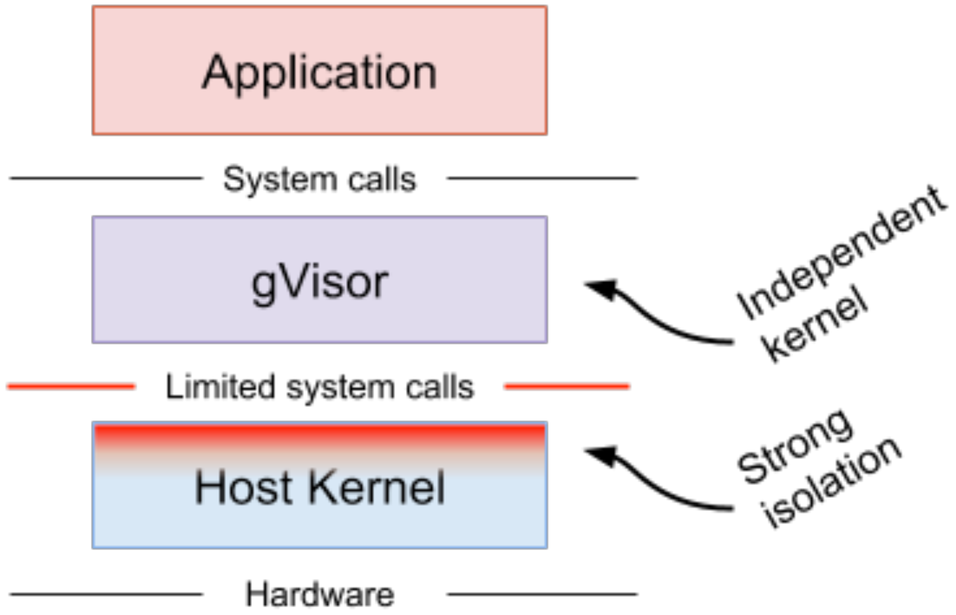
バイナリの setuid root	バイナリの権限が大きく、 さらなる制御をするための対応が必要
LXC方式の非特権コンテナ	一部バイナリをsetuid rootするので コンテナの機能の変化で追加など対応が必要
Rootless runc	ネットワークがtapのみなど 機能に制限がある

Hypervisor等の利用（予稿では未言及）

- コンテナ技術のセキュリティ機構の多くは、OSの中で閉じているが、

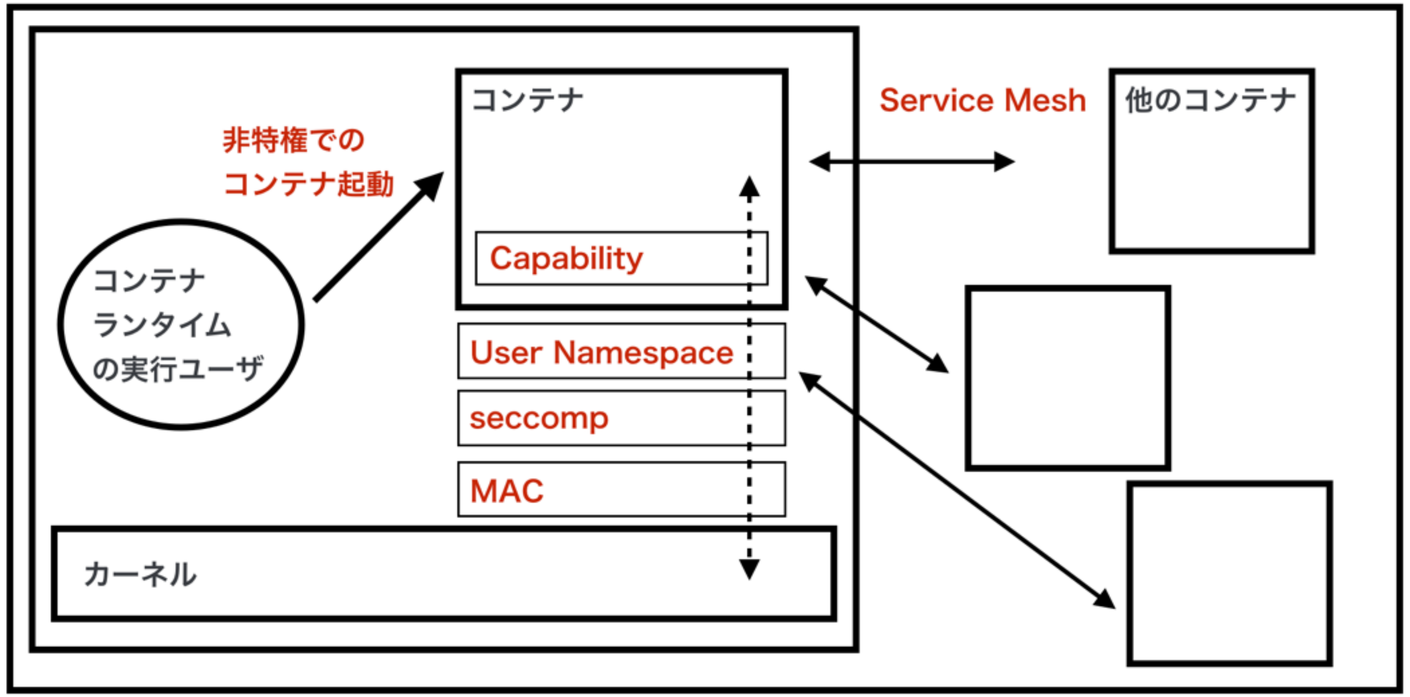
  軽量なハイパーバイザなどを利用し、その範囲を超えたサンドボックス化を実現する技術が現れている

- gVisor のREADMEより:



位置付ける

- 課題1./3.はカーネル/コンテナランタイムで手厚くカバー
- 課題2.はService Meshの進化で
- 課題4.は非特権コンテナが
  一定で有効か



課題への対応

機能名	対応可能な課題
User Namespace	1, 2, (3).
Kernel Capabilities	1, 2, 3.
seccomp	1, 2, 3.
Mandatory Access Control	2, 3, (4).
Service Mesh	2
非特権コンテナ	4
ハイパーバイザの利用	1, 2, 3

課題4. への対応は限定的

機能名	対応可能な課題
User Namespace	1, 2, (3).
Kernel Capabilities	1, 2, 3.
seccomp	1, 2, 3.
Mandatory Access Control	2, 3, (4).
Service Mesh	2
非特権コンテナ	4
ハイパーバイザの利用	1, 2, 3

3.

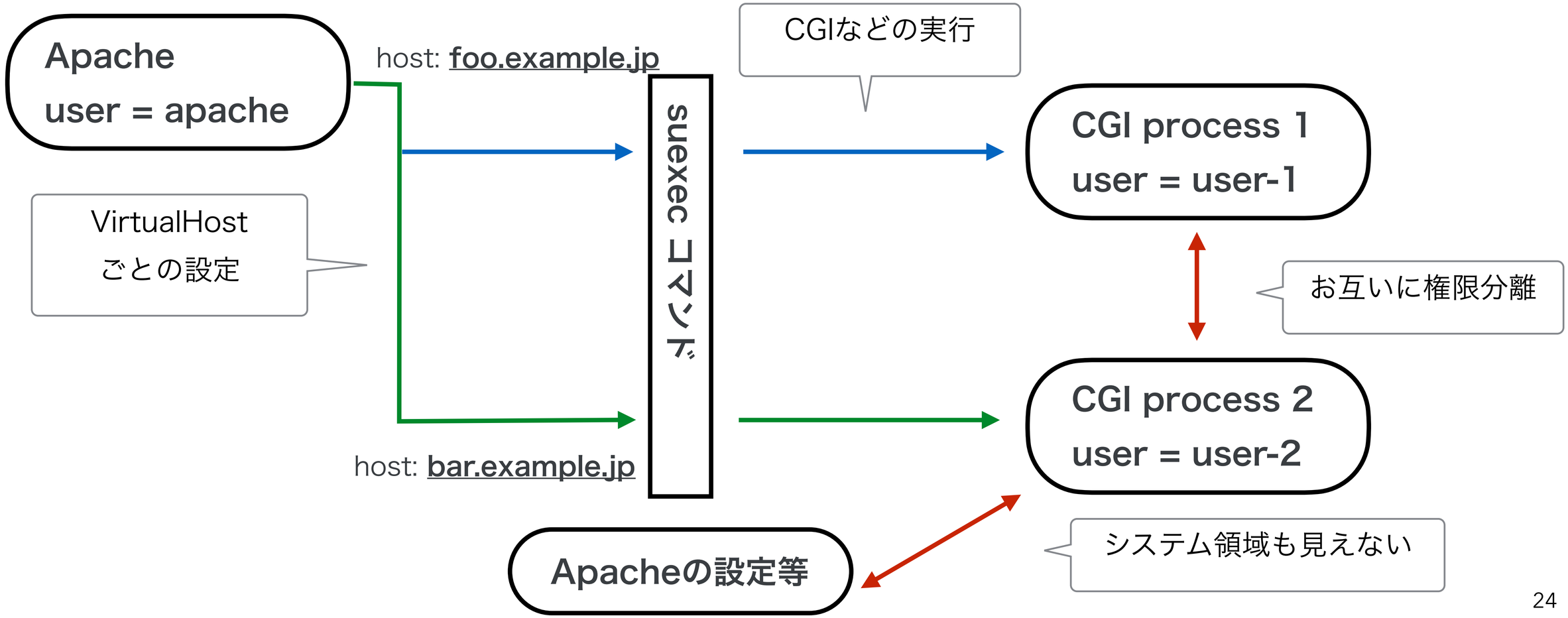
提案するバリデーション機構について

提案: suEXECセキュリティモデルベースの検査導入

- suEXECセキュリティモデル:
  - Apache HTTPサーバがCGIあるいはSSIを実行する際に、Apache自体のユーザ ID とは 異なるユーザ ID で実行する際に考慮しているモデル
- 課題4. へのさらなる対応に主眼を置く

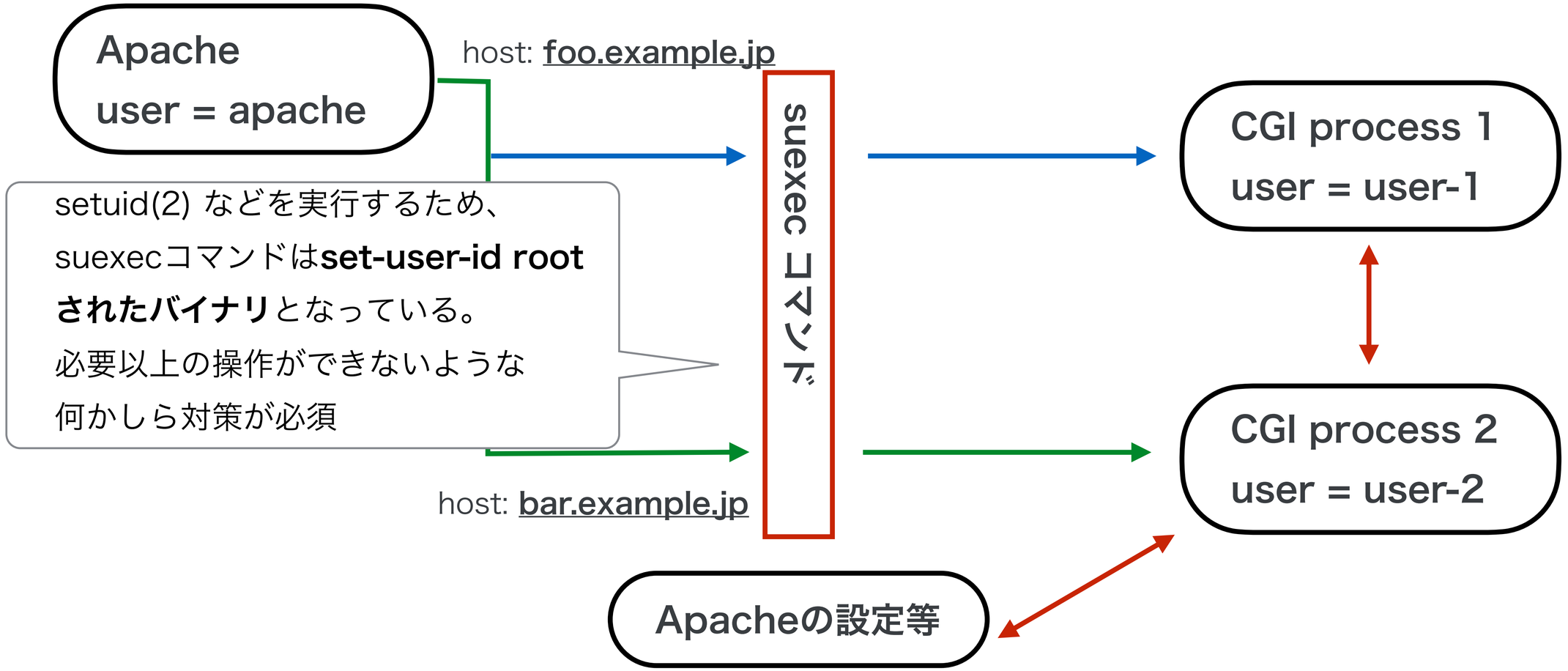
suEXECセキュリティモデルとは

- まず、suEXECについて



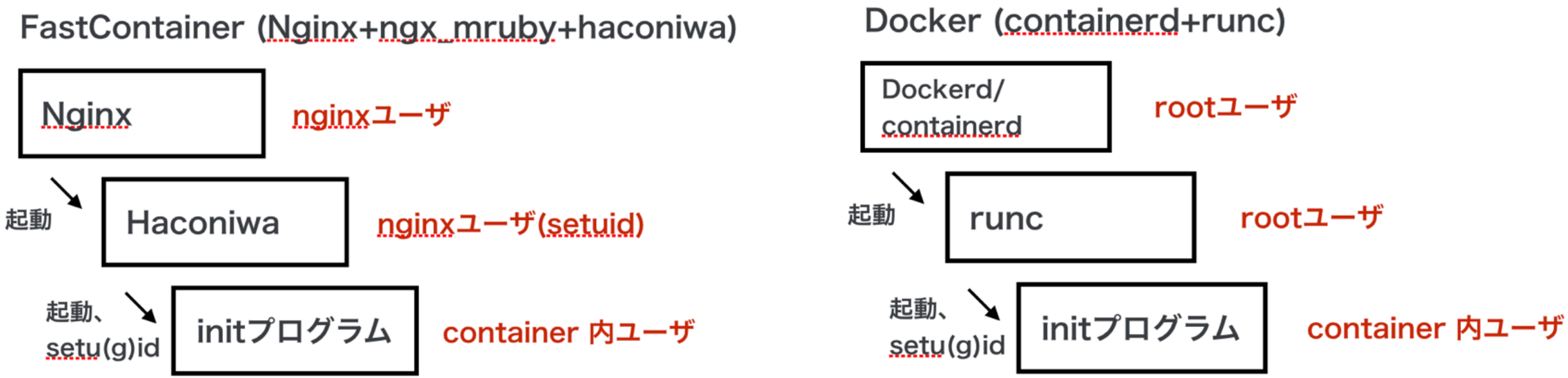
suEXECセキュリティモデルが必要な理由

- set-user-id root されたバイナリを経由して実現していることへの対応



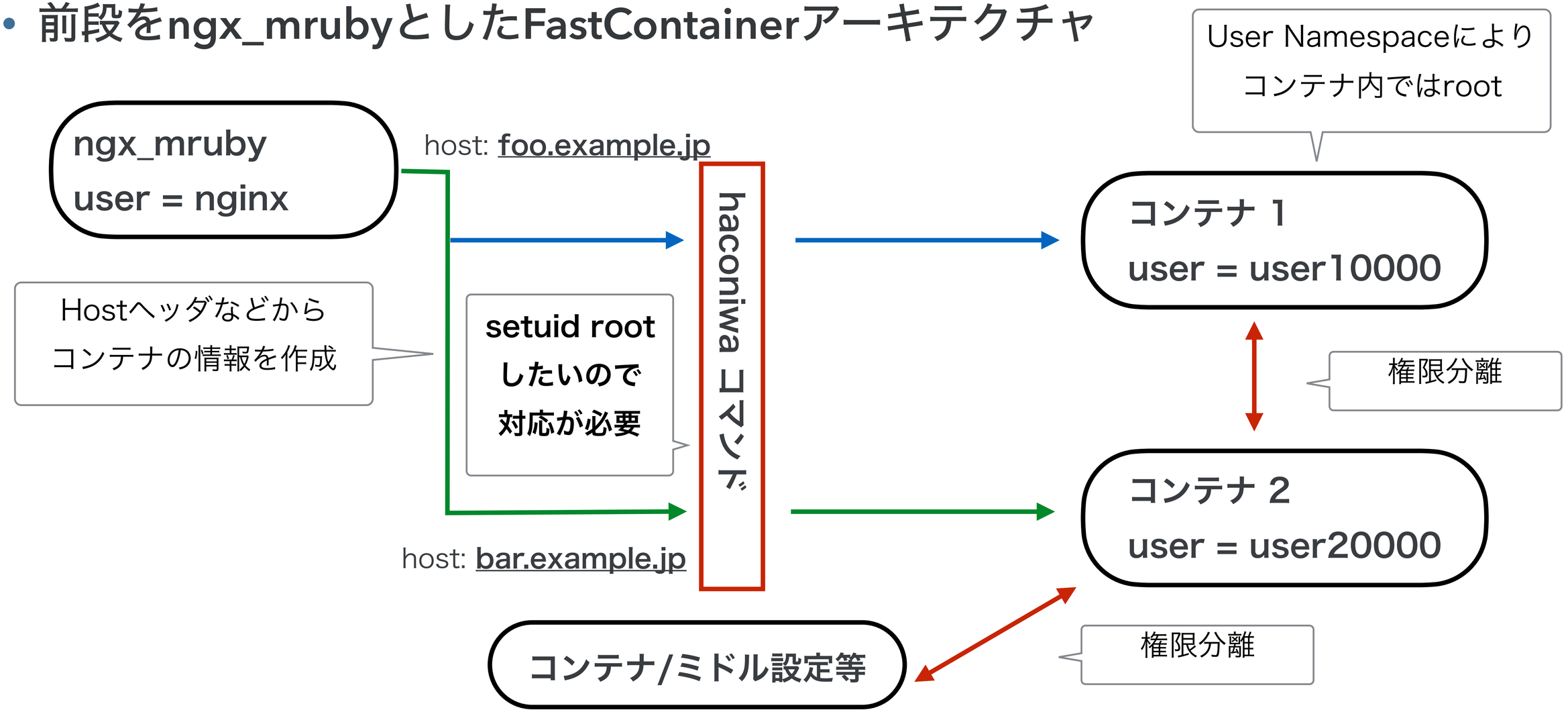
コンテナイベントを受け取る前段サービス

- コンテナには、起動イベントを受け取る前段サービスがある
- FastContainerのngx_mruby、runcに対するcontainerd、など



コンテナの前段プログラムと比較する

- 前段をngx_mrubyとしたFastContainerアーキテクチャ



suEXECセキュリティモデルの詳細

- 特徴としては
  - 必要以上の権限・参照を与えない
  - UNIXユーザベース
  - 不正なIDなどは厳しく検査

1. **wrapper を実行しているユーザはこのシステムの正当なユーザか?**

   これは、wrapper を実行しているユーザが本当にシステムの利用者であることを保証するためです。

2. **wrapper が適切な数の引数で呼び出されたか?**

   wrapper は適切な数の引数が与えられた場合にのみ実行されます。適切な引数のフォーマットは Apache Web サーバに解
   なければ、攻撃をされたかあなたの Apache バイナリの suEXEC の部分がどこかおかしい可能性があります。

3. **この正当なユーザは wrapper の実行を許可されているか?**

   このユーザは wrapper 実行を許可されたユーザですか? ただ一人のユーザ (Apache ユーザ) だけが、このプログラムの実

4. **対象の CGI, SSI プログラムが安全でない階層の参照をしているか?**

   対象の CGI, SSI プログラムが '/' から始まる、または '..' による参照を行なっていますか? これらは許可されません。対象
   トルート (下記の `--with-suexec-docroot=`*`DIR`* を参照) 内に存在しなければなりません。

5. **対象となるユーザ名は正当なものか?**

   対象となるユーザ名は存在していますか?

6. **対象となるグループ名は正当なものか?**

   対象となるグループ名は存在していますか?

7. **目的のユーザはスーパーユーザではないか?**

コンテナの起動プロセスに対応する

- wrapper = コンテナランタイム
- wrapper を実行しているユーザ = 前段サービスの実行ユーザ
- 対象の CGI、SSIプログラム = コンテナのPID=1プロセス(initと呼ぶ)
- 対象となるユーザ ID, ... = コンテナのinitの実行ユーザ
- CGI/SSI プログラムのドキュメントディレクトリ = コンテナの rootfs

元の分類を3パターンに整理

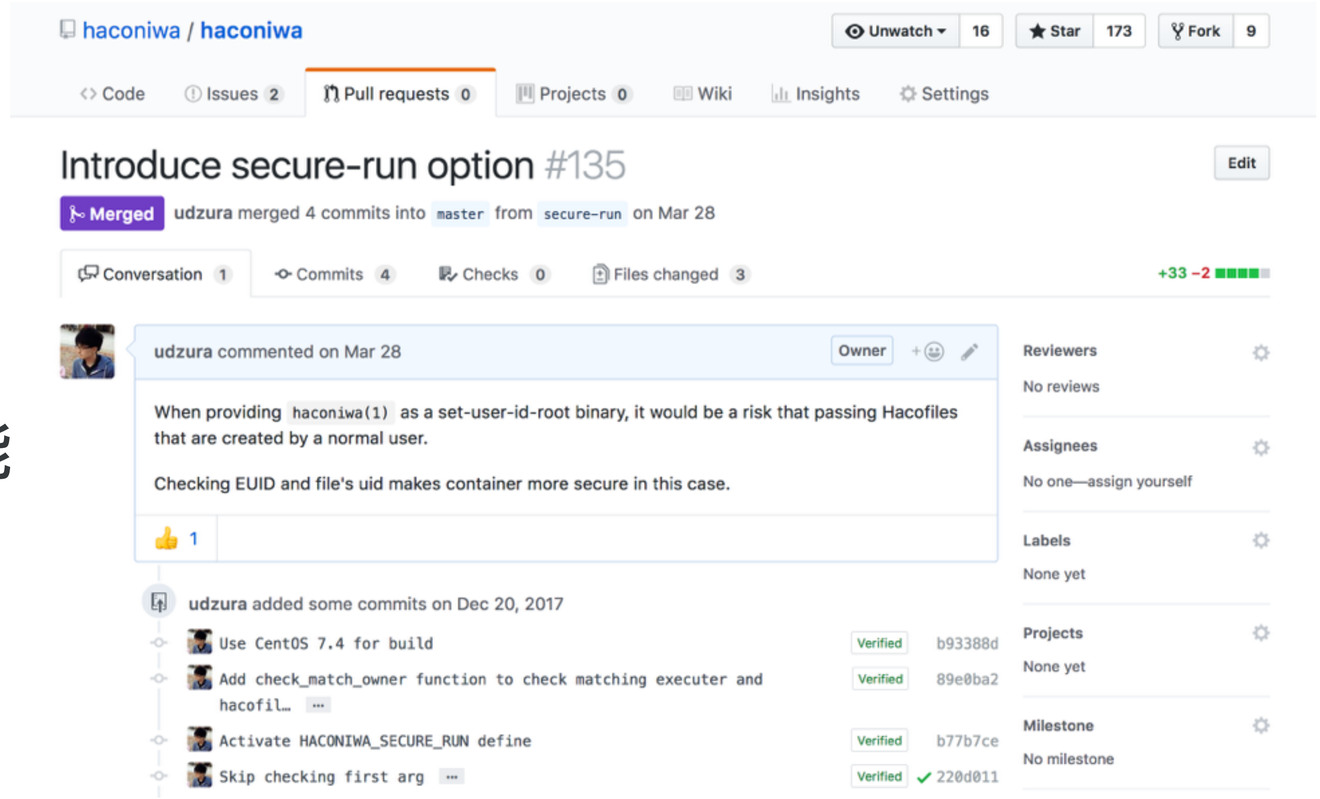
- バリデーションとして選択的に処理すべきもの
  - 設定とバイナリのオウナー一致、rootfsの位置、initを実行するユーザIDに関する制限、環境変数の検査などを行う（9ルール）
- エラーとして処理すべきもの
  - IDの存在確認など致命的なものは必ず検査する（5ルール）
- コンテナの運用の性質上、実装すべきかどうか検討が必要なもの
  - コンテナの機能と単純な対応関係にない・単純対応をすると運用の問題が起きそうなものは今回保留する（2ルール）

実装の詳細な方針など補遺

- 「エラーとして処理すべきもの」としたルールは適切なエラーメッセージを出すようにする
- 選択的に処理すべきものについては、ビルド時オプションでどの検査を実施するかを指定可能とする。オプションが多いほどセキュアだが作成できるコンテナに制限がある。少ないと逆の性質となる。
- ビルド時オプションにしたのは、外部の設定では挙動を変えないバイナリ自体のオプションとし、攻撃の機会を減らすという意図

具体的実装

- 筆者の開発するコンテナランタイム Haconiwa で順次実装する予定
- 「コンテナランタイムの設定ファイルと、コンテナランタイム自身のオウナーが一致するか」の検査を実装済
- HaconiwaはmrubyのビルドDSL（Ruby DSL）を利用でき柔軟にビルドオプション設定可能



4.

課題・議論したいこと

課題

- 具体的実装への落とし込み
  - これは順次作っていくばかりである
- 「実装すべきかどうか検討が必要なもの」の検討
  - そもそも実装しなくていいのか、守っているもののエッセンスは何か
- 導入後の問題点、有効性の検証
  - パフォーマンス、実際に攻撃を防げるか、運用要件を満たすかetc

議論したいところ

1. suEXEC モデルをヒントにしたこれらの検査は本当にセキュアになるのか？

   A. コンテナの起動と一般のWebサーバのプロセス起動を比較しているが、
      妥当なのかどうか

   B. FastContainer のような場合のみなのか、一般に有効なのか

2. 一部、元のモデルでそのまま適用できなそうなものがあると論じたが、
   それらの扱いをどうすると良いか？

3. 運用に乗せた際の効果測定にはどのような方法が考えられるか？