
Container Security: What Could Possibly Go Wrong?

— Michaela Doležalová —
Daniel Kouřil

Masaryk University, CESNET

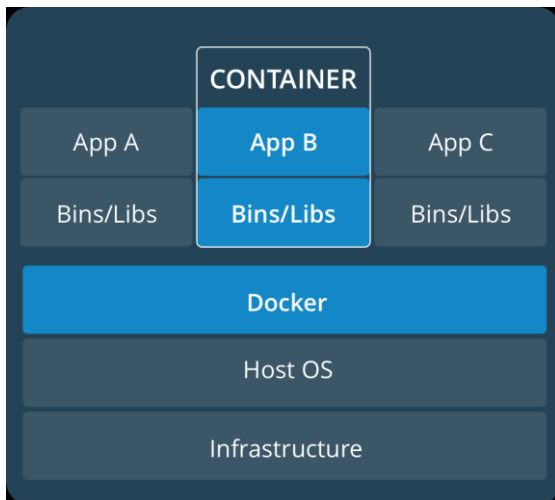
What is a container?

- fundamentally, a container is just **a running process**
- it is **isolated** from the host and from other containers
- each container usually interacts with its **own private filesystem**
- there are different containerization technologies available
(Docker, LXD, Podman, Singularity, ...)
- in this tutorial, we will focus mainly on Docker

Containers vs. Virtual Machines

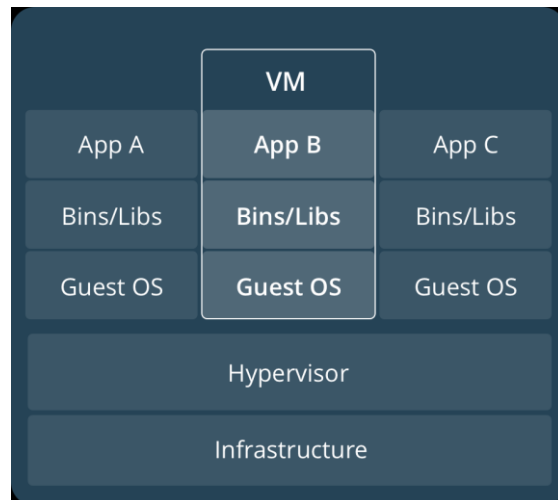
- a container is **an abstraction of the application layer**

(it runs natively on Linux)



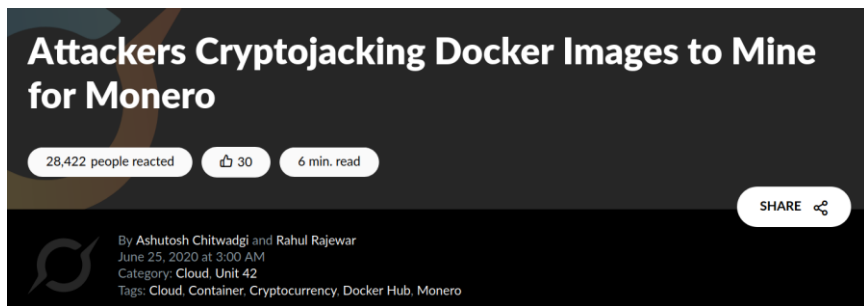
- a virtual machine is **an abstraction of the hardware layer**

(it runs a full-blown “guest” operating system)



Threat Landscape

- proper **deployment** and **configuration** requires understanding the technology
- **image management** (integrity and authenticity of the image)
- trust in the **image maintainer** and the **repository operator**
- **malicious images** may be found even in an official registry



<https://unit42.paloaltonetworks.com/cryptojacking-docker-images-for-mining-monero/>

Usual Best Practice

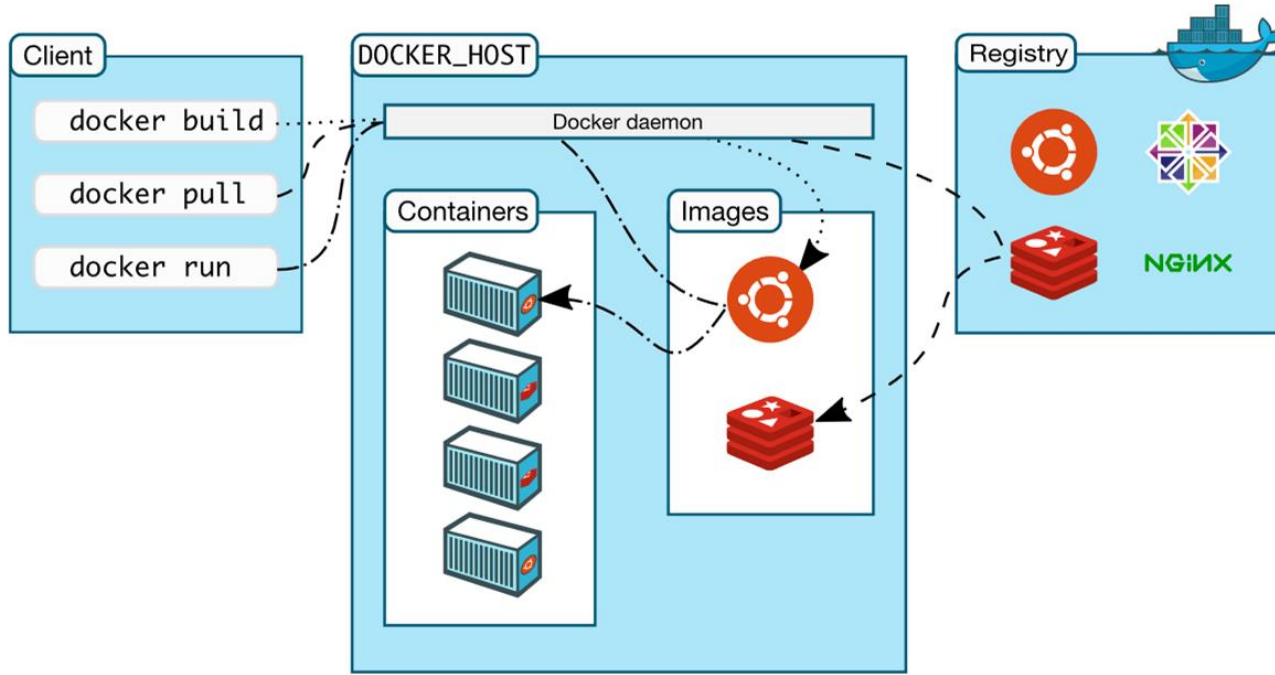
- especially proper **vulnerability/patch management**
- it is often kernel-related and therefore requiring reboot
- updates **not always** available
- **extremely important** (couple of vulns over the past few years)
- out of scope for today

Let's move to Docker itself....

Docker Terminology

- **Docker container image** - a lightweight, standalone, executable package of software that includes everything needed to run an application
(code, runtime, system tools, system libraries and settings)
- an image is usually pulled from a **registry** to a host machine
*(e.g. **DockerHub** - something like a Google Play store, Apple store, etc.)*
- **Docker container** - an instance of an image
- a host machine runs the **container engine (Docker Daemon)**

Docker Architecture



Docker Container Creation

- the image is opened up and the **filesystem** of that image is copied into a **temporary archive** on the host
 - when removed, any changes to its state **disappear**
- the container engine manages the process tree **natively** on the kernel
- to provide application sandboxing, Docker uses Linux **namespaces** and **cgroups**
- when you start a container with *docker run*, Docker creates **a set of namespaces** and **control groups**

Namespaces

- Docker Engine uses the following namespaces on Linux
 - **PID namespace** for process isolation
 - **NET namespace** for managing/separating network interfaces
 - **IPC namespace** for separating inter-process communication
 - **MNT namespace** for managing/separating filesystem mount points
 - **UTS namespace** for isolating kernel and version identifiers
(mainly to set the hostname and domainname visible to the process)
 - **User ID (user) namespace** for privilege isolation
- user namespace **must be enabled** on purpose, it is **not** used by default

PID namespace

- allows to establish **separate process trees**
- the complete picture still **visible** from the **host** (outside the namespace)

1029	?	Ssl	7:48	/usr/bin/containerd
28834	?	Sl	0:00	_ containerd-shim -namespace moby
28851	pts/0	Ss	0:00	_ bash
28899	pts/0	S+	0:00	_ dash

```
root# docker run --rm -it debian/ps bash
root@3146c2faec9b:/# dash
# ps af
```

PID	TTY	STAT	TIME	COMMAND
1	pts/0	Ss	0:00	bash
6	pts/0	S	0:00	dash
7	pts/0	R+	0:00	_ ps af

User ID (user) Namespace

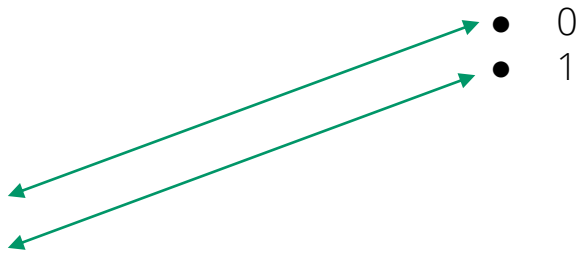
- enables **different uid/gid** structures **visible** to the **kernel**
- **mapping** between uids in the namespace and “global” uids is **needed**
- by default, **root in the container is root in the host** !

global (host) id's

- 0
- 1
-
- 1000
- 1001
- ...
- 100000
- 100001

id's in the namespace

- 0
- 1



Cgroups

- short for **control groups**
- they allow Docker Engine to **share available hardware resources**
- they help to ensure that a single container cannot bring the system down
- they implement **resource accounting and limiting** (CPU, disk I/O, etc.)

Linux Kernel Capabilities

- capabilities turn the binary “root/non-root” dichotomy into a **fine-grained access control system**
- by default, Docker starts containers with **a restricted set of capabilities**
- Docker supports the **addition** and **removal** of capabilities
- additional capabilities extends the utility but has security implications, too
- a container started with **--privileged flag** obtains **all** capabilities
- running **without --privileged** doesn't mean the container doesn't have root privileges!

I am root. Or not?

- multiple levels of root privileges, from an unprivileged root user:
 - if user namespace is **enabled**, root inside a container has no root privileges outside in the host system
 - **by default**, root in a container has some privileges
 - but these are restricted by the **default set of capabilities**
 - we can **explicitly** add **extra capabilities** to our root in a container
 - with the **--privileged flag**, we have full root rights granted



root

```
root# docker run --rm -it debian/ip bash
root@b523a39fcc48:/# iptables -L -n
iptables: Permission denied (you must be root).
root@b523a39fcc48:/#
```



root

```
root# docker run --rm -it --cap-add=NET_ADMIN debian/ip bash
root@361c51aa11b0:/# iptables -L -n
Chain INPUT (policy ACCEPT)
target      prot opt source                destination

Chain FORWARD (policy ACCEPT)
target      prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target      prot opt source                destination
root@361c51aa11b0:/#
```

Docker Daemon

- running containers (and applications) with Docker implies running the Docker daemon
- to control it, it requires **root privileges**, or **docker group membership**
- only **trusted users** should be allowed to control your Docker daemon
- it allows you to share a directory between the Docker host and a guest container
- e.g. we can start a container where the /host directory is the / directory on your host

Docker API

- an **API** for interacting with the **Docker daemon**
- **by default**, the Docker daemon listens for Docker API requests at a unix domain socket created at **/var/run/docker.sock**
- with -H it is possible to make the Docker daemon listen on a specific IP and port
- you **could** set it to 0.0.0.0:2375 or a specific host IP to give access to everybody
- Docker API requests go, by default, to the **Docker daemon of the host**

Docker vs. chroot command

- a container **isn't instantiated by the user** but the Docker daemon!
- anyone who's allowed to communicate with the Docker daemon **can manage containers**
- that includes using any **configuration parameters**
- they can play with binding/mounting files/directories
- or decide which user id will be used in the container
 - including root (unlike eg. chroot) !

Examples of Docker-related incidents

- **unprotected access** to Docker daemon over the Internet
 - revealed by common Internet scans
 - instantiation of malicious containers used for dDoS activities
- **stolen credentials** providing access to the Docker daemon
 - used to deploy a container set up in a way allowing breaking the isolation
 - the attackers escaped to the host system
 - an deployed crypto-mining software and misused the resources

Other kernel security features

- it is possible to **enhance Docker security** with systems like TOMOYO, AppArmor, SELinux, etc.
- you can also run the kernel with GRSEC and PAX
- all these extra security features require **extra effort**
- some of them are **only for containers** and not for the Docker daemon
- as of Docker 1.10 User Namespaces are **supported directly** by the Docker daemon

Summary and Cheat Sheets

Docker Cheat Sheet - Running a Container

start a new container from an image

`docker run IMAGE`

start a new container from an image and assign it a name

`docker run --name IMAGE`

start a new container from an image and map a port

`docker run -p HOSTPORT:CONTAINERPORT IMAGE`

start a new container in background

`docker run -d IMAGE`

start a new container and assign it a hostname

`docker run --hostname HOSTNAME IMAGE`

start a new container and map a local directory into the container

`docker run -v HOSTDIR:TARGETDIR IMAGE`

Docker Cheat Sheet - Managing a Container

show a list of running containers

stop

a

running container

`docker ps`

`docker stop CONTAINER`

show a list of all containers

start a stopped container

`docker ps -a`

`docker start CONTAINER`

delete a container

copy a file from a container to the host

`docker rm CONTAINER`

`docker cp CONTAINER:SOURCE TARGET`

delete a running container

copy a file from the host to a container

`docker rm -f CONTAINER`

Docker Cheat Sheet - Managing Images

download an image

docker pull IMAGE

upload an image to a repository

docker push IMAGE

build an image from a Dockerfile

docker build DIRECTORY

Docker Cheat Sheet - Info and Stats

show the logs of a container

docker logs CONTAINER

show stats of running containers

docker stats

show processes of a container

docker top CONTAINER

show installed docker version

docker version

Practical Part

Cyber Range KYPO

- platform to organize and control cyber exercise, mostly CTF-like events
- set of services on the top of OpenStack cloud, providing separated *sandboxes*
 - machines are instantiated as VMs, connected using isolated network
- web portal mediating access to the environment and guiding participants through levels
 - description, tasks, hints
 - levels are linked using flags
- scoreboard and monitoring of progress for organizers
- platform is open-source, actively maintained by Masaryk University
 - <https://kypo.muni.cz/>

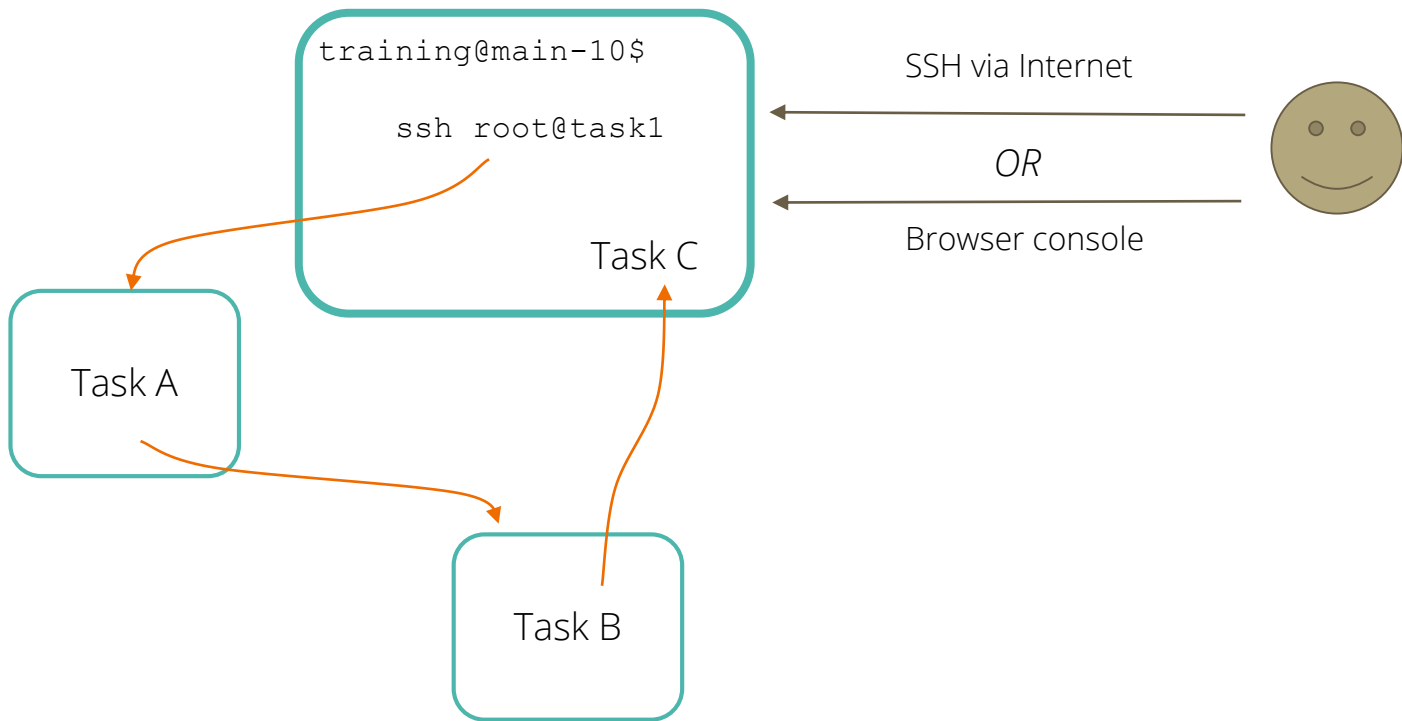
How To Get Started

- “book” your account at
 - https://docs.google.com/spreadsheets/d/1qlZB_SPjXlMwePs2H9yGaBmTiVWDwp_sTq4Czl7oi_e4/
- log in portal <https://iris.crp.kypo.muni.cz/> using the booked credentials
 - you will start off the intro page
 - 14 “levels” in total (inc. intro etc.), each level contains
 - description
 - hints
 - specification of the flag
 - once you determine the flag, submit it to get to the next level
- interaction with VMs via either
 - embedded console (see the topology, click the “main” node (right mouse button) and open the console
 - directly using SSH (but ignore the “Get SSH Access”)

SSH access

- don't use the "Get SSH Access" button
 - works but it's complicated
- connect to the machine using:
 - host: **iris.crp.kypo.muni.cz**
 - port: as given in the sheet with credentials
 - user: **training**
 - password: **20202020**
 - e.g. `ssh -p 5003 training@iris.crp.kypo.muni.cz`
- you'll land on the "main" host (the same can be accessed via browser console)

Topology



Training Run Overview

Access Training

Fill in access token provided by the organizer to start the training



Access Token Prefix *

Access Token PIN *

Access

run 1 or run 2

Resume training run

Title	Date	Completed Levels	Actions
Docker Security Training - run 1	10 Feb 2021 10:39 - 12 Feb 2021 23:59	1/14	 

Items per page: 10 1 - 1 of 1 < >



Demo User3

user3@oidc.csirt.muni.cz

01:34:31

Docker Security Training

Welcome to this short Docker Security Training. There are three tasks which will illustrate common misconceptions regarding Docker.

The first task (Task A) is divided into several levels to provide you with smooth passage through the training. There is a description of your task at each level and also the description of the flag you need to submit to get to the next level. If you want, you can have hints revealed to you how to finish the task. In case you have no idea what to do, you can have the whole solution revealed.

There will be a short summary after the Task A. Then, you will be given a flag to start with Tasks B and C.

We encourage you to search a bit on the Internet in case you need. Have fun!

[Next](#)

Thank you for your attention.

Please be so kind and fill in our short questionnaire:

<https://forms.gle/4gbuJyD3ZLDRFRHY8>