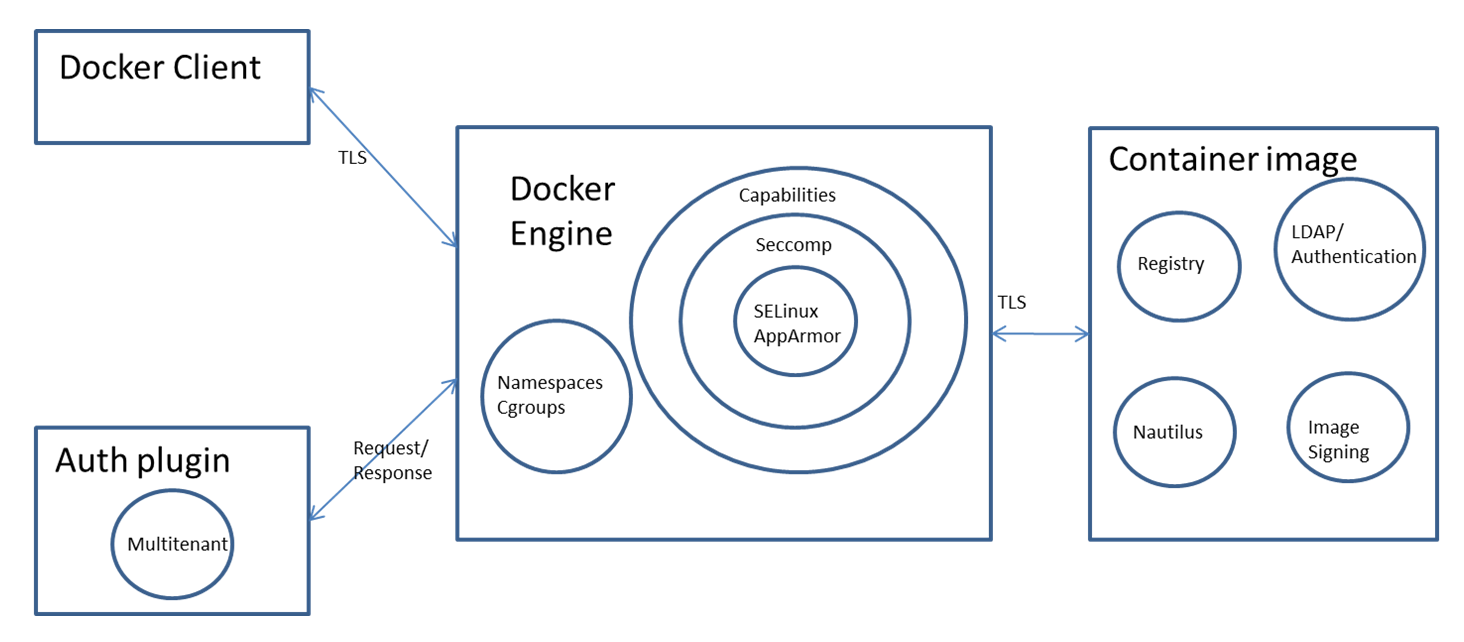
**Basic Docker Engine Security**

The **Docker engine** employs the Linux kernel's **Namespaces** and **Cgroups** to isolate containers, offering a basic layer of security. Additional protection is provided through **Capabilities dropping**, **Seccomp**, and **SELinux/AppArmor**, enhancing container isolation. An **auth plugin** can further restrict user actions.



Docker Security

**Secure Access to Docker Engine**

The Docker engine can be accessed either locally via a Unix socket or remotely using HTTP. For remote access, it's essential to employ HTTPS and **TLS** to ensure confidentiality, integrity, and authentication.

The Docker engine, by default, listens on the Unix socket at unix:///var/run/docker.sock. On Ubuntu systems, Docker's startup options are defined in /etc/default/docker. To enable remote access to the Docker API and client, expose the Docker daemon over an HTTP socket by adding the following settings:

Copy

DOCKER\_OPTS="-D -H unix:///var/run/docker.sock -H tcp://192.168.56.101:2376"

sudo service docker restart

However, exposing the Docker daemon over HTTP is not recommended due to security concerns. It's advisable to secure connections using HTTPS. There are two main approaches to securing the connection:

1. The client verifies the server's identity.
2. Both the client and server mutually authenticate each other's identity.

Certificates are utilized to confirm a server's identity. For detailed examples of both methods, refer to [**this guide**](https://sreeninet.wordpress.com/2016/03/06/docker-security-part-3engine-access/).

**Security of Container Images**

Container images can be stored in either private or public repositories. Docker offers several storage options for container images:

* [**Docker Hub**](https://hub.docker.com/): A public registry service from Docker.
* [**Docker Registry**](https://github.com/docker/distribution): An open-source project allowing users to host their own registry.
* [**Docker Trusted Registry**](https://www.docker.com/docker-trusted-registry): Docker's commercial registry offering, featuring role-based user authentication and integration with LDAP directory services.

**Image Scanning**

Containers can have **security vulnerabilities** either because of the base image or because of the software installed on top of the base image. Docker is working on a project called **Nautilus** that does security scan of Containers and lists the vulnerabilities. Nautilus works by comparing the each Container image layer with vulnerability repository to identify security holes.

For more [**information read this**](https://docs.docker.com/engine/scan/).

* **docker scan**

The **docker scan** command allows you to scan existing Docker images using the image name or ID. For example, run the following command to scan the hello-world image:

Copy

docker scan hello-world

Testing hello-world...

Organization: docker-desktop-test

Package manager: linux

Project name: docker-image|hello-world

Docker image: hello-world

Licenses: enabled

✓ Tested 0 dependencies for known issues, no vulnerable paths found.

Note that we do not currently have vulnerability data for your image.

* [**trivy**](https://github.com/aquasecurity/trivy)

Copy

trivy -q -f json <container\_name>:<tag>

* [**snyk**](https://docs.snyk.io/snyk-cli/getting-started-with-the-cli)

Copy

snyk container test <image> --json-file-output=<output file> --severity-threshold=high

* [**clair-scanner**](https://github.com/arminc/clair-scanner)

Copy

clair-scanner -w example-alpine.yaml --ip YOUR\_LOCAL\_IP alpine:3.5

**Docker Image Signing**

Docker image signing ensures the security and integrity of images used in containers. Here's a condensed explanation:

* **Docker Content Trust** utilizes the Notary project, based on The Update Framework (TUF), to manage image signing. For more info, see [Notary](https://github.com/docker/notary) and [TUF](https://theupdateframework.github.io/).
* To activate Docker content trust, set export DOCKER\_CONTENT\_TRUST=1. This feature is off by default in Docker version 1.10 and later.
* With this feature enabled, only signed images can be downloaded. Initial image push requires setting passphrases for the root and tagging keys, with Docker also supporting Yubikey for enhanced security. More details can be found [here](https://blog.docker.com/2015/11/docker-content-trust-yubikey/).
* Attempting to pull an unsigned image with content trust enabled results in a "No trust data for latest" error.
* For image pushes after the first, Docker asks for the repository key's passphrase to sign the image.

To back up your private keys, use the command:

Copy

tar -zcvf private\_keys\_backup.tar.gz ~/.docker/trust/private

When switching Docker hosts, it's necessary to move the root and repository keys to maintain operations.

**Containers Security Features**

Summary of Container Security Features

**Main Process Isolation Features**

In containerized environments, isolating projects and their processes is paramount for security and resource management. Here's a simplified explanation of key concepts:

**Namespaces**

* **Purpose**: Ensure isolation of resources like processes, network, and filesystems. Particularly in Docker, namespaces keep a container's processes separate from the host and other containers.
* **Usage of unshare**: The unshare command (or the underlying syscall) is utilized to create new namespaces, providing an added layer of isolation. However, while Kubernetes doesn't inherently block this, Docker does.
* **Limitation**: Creating new namespaces doesn't allow a process to revert to the host's default namespaces. To penetrate the host namespaces, one would typically require access to the host's /proc directory, using nsenter for entry.

**Control Groups (CGroups)**

* **Function**: Primarily used for allocating resources among processes.
* **Security Aspect**: CGroups themselves don't offer isolation security, except for the release\_agent feature, which, if misconfigured, could potentially be exploited for unauthorized access.

**Capability Drop**

* **Importance**: It's a crucial security feature for process isolation.
* **Functionality**: It restricts the actions a root process can perform by dropping certain capabilities. Even if a process runs with root privileges, lacking the necessary capabilities prevents it from executing privileged actions, as the syscalls will fail due to insufficient permissions.

These are the **remaining capabilities** after the process drop the others:

Copy

Current: cap\_chown,cap\_dac\_override,cap\_fowner,cap\_fsetid,cap\_kill,cap\_setgid,cap\_setuid,cap\_setpcap,cap\_net\_bind\_service,cap\_net\_raw,cap\_sys\_chroot,cap\_mknod,cap\_audit\_write,cap\_setfcap=ep

**Seccomp**

It's enabled by default in Docker. It helps to **limit even more the syscalls** that the process can call. The **default Docker Seccomp profile** can be found in <https://github.com/moby/moby/blob/master/profiles/seccomp/default.json>

**AppArmor**

Docker has a template that you can activate: <https://github.com/moby/moby/tree/master/profiles/apparmor>

This will allow to reduce capabilities, syscalls, access to files and folders...

### Namespaces

**Namespaces** are a feature of the Linux kernel that **partitions kernel resources** such that one set of **processes** **sees** one set of **resources** while **another** set of **processes** sees a **different** set of resources. The feature works by having the same namespace for a set of resources and processes, but those namespaces refer to distinct resources. Resources may exist in multiple spaces.

Docker makes use of the following Linux kernel Namespaces to achieve Container isolation:

* pid namespace
* mount namespace
* network namespace
* ipc namespace
* UTS namespace

For **more information about the namespaces** check the following page: Open word file of Namespaces

### cgroups

Linux kernel feature **cgroups** provides capability to **restrict resources like cpu, memory, io, network bandwidth among** a set of processes. Docker allows to create Containers using cgroup feature which allows for resource control for the specific Container. Following is a Container created with user space memory limited to 500m, kernel memory limited to 50m, cpu share to 512, blkioweight to 400. CPU share is a ratio that controls Container’s CPU usage. It has a default value of 1024 and range between 0 and 1024. If three Containers have the same CPU share of 1024, each Container can take upto 33% of CPU in case of CPU resource contention. blkio-weight is a ratio that controls Container’s IO. It has a default value of 500 and range between 10 and 1000.

Copy

docker run -it -m 500M --kernel-memory 50M --cpu-shares 512 --blkio-weight 400 --name ubuntu1 ubuntu bash

To get the cgroup of a container you can do:

Copy

docker run -dt --rm denial sleep 1234 #Run a large sleep inside a Debian container

ps -ef | grep 1234 #Get info about the sleep process

ls -l /proc/<PID>/ns #Get the Group and the namespaces (some may be uniq to the hosts and some may be shred with it)

For more information check: Open word file of Namespaces

Capabilities

Capabilities allow **finer control for the capabilities that can be allowed** for root user. Docker uses the Linux kernel capability feature to **limit the operations that can be done inside a Container** irrespective of the type of user.

When a docker container is run, the **process drops sensitive capabilities that the proccess could use to escape from the isolation**. This try to assure that the proccess won't be able to perform sensitive actions and escape:

Open word file of Linux Capabilities

### Seccomp in Docker

This is a security feature that allows Docker to **limit the syscalls** that can be used inside the container:

[PAGESeccomp](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security/seccomp)

### AppArmor in Docker

**AppArmor** is a kernel enhancement to confine **containers** to a **limited** set of **resources** with **per-program profiles**.:

[PAGEAppArmor](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security/apparmor)

### SELinux in Docker

* **Labeling System**: SELinux assigns a unique label to every process and filesystem object.
* **Policy Enforcement**: It enforces security policies that define what actions a process label can perform on other labels within the system.
* **Container Process Labels**: When container engines initiate container processes, they are typically assigned a confined SELinux label, commonly container\_t.
* **File Labeling within Containers**: Files within the container are usually labeled as container\_file\_t.
* **Policy Rules**: The SELinux policy primarily ensures that processes with the container\_t label can only interact (read, write, execute) with files labeled as container\_file\_t.

This mechanism ensures that even if a process within a container is compromised, it's confined to interacting only with objects that have the corresponding labels, significantly limiting the potential damage from such compromises.

[PAGESELinux](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/selinux)

### AuthZ & AuthN

In Docker, an authorization plugin plays a crucial role in security by deciding whether to allow or block requests to the Docker daemon. This decision is made by examining two key contexts:

* **Authentication Context**: This includes comprehensive information about the user, such as who they are and how they've authenticated themselves.
* **Command Context**: This comprises all pertinent data related to the request being made.

These contexts help ensure that only legitimate requests from authenticated users are processed, enhancing the security of Docker operations.

[PAGEAuthZ& AuthN - Docker Access Authorization Plugin](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security/authz-and-authn-docker-access-authorization-plugin)

## DoS from a container

If you are not properly limiting the resources a container can use, a compromised container could DoS the host where it's running.

* CPU DoS

Copy

# stress-ng

sudo apt-get install -y stress-ng && stress-ng --vm 1 --vm-bytes 1G --verify -t 5m

# While loop

docker run -d --name malicious-container -c 512 busybox sh -c 'while true; do :; done'

* Bandwidth DoS

Copy

nc -lvp 4444 >/dev/null & while true; do cat /dev/urandom | nc <target IP> 4444; done

## Interesting Docker Flags

### --privileged flag

In the following page you can learn **what does the --privileged flag imply**:

[PAGEDocker --privileged](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security/docker-privileged)

### --security-opt

#### no-new-privileges

If you are running a container where an attacker manages to get access as a low privilege user. If you have a **miss-configured suid binary**, the attacker may abuse it and **escalate privileges inside** the container. Which, may allow him to escape from it.

Running the container with the **no-new-privileges** option enabled will **prevent this kind of privilege escalation**.

Copy

docker run -it --security-opt=no-new-privileges:true nonewpriv

#### Other

Copy

#You can manually add/drop capabilities with

--cap-add

--cap-drop

# You can manually disable seccomp in docker with

--security-opt seccomp=unconfined

# You can manually disable seccomp in docker with

--security-opt apparmor=unconfined

# You can manually disable selinux in docker with

--security-opt label:disable

For more **--security-opt** options check: <https://docs.docker.com/engine/reference/run/#security-configuration>

## Other Security Considerations

### Managing Secrets: Best Practices

It's crucial to avoid embedding secrets directly in Docker images or using environment variables, as these methods expose your sensitive information to anyone with access to the container through commands like docker inspect or exec.

**Docker volumes** are a safer alternative, recommended for accessing sensitive information. They can be utilized as a temporary filesystem in memory, mitigating the risks associated with docker inspect and logging. However, root users and those with exec access to the container might still access the secrets.

**Docker secrets** offer an even more secure method for handling sensitive information. For instances requiring secrets during the image build phase, **BuildKit** presents an efficient solution with support for build-time secrets, enhancing build speed and providing additional features.

To leverage BuildKit, it can be activated in three ways:

1. Through an environment variable: export DOCKER\_BUILDKIT=1
2. By prefixing commands: DOCKER\_BUILDKIT=1 docker build .
3. By enabling it by default in the Docker configuration: { "features": { "buildkit": true } }, followed by a Docker restart.

BuildKit allows for the use of build-time secrets with the --secret option, ensuring these secrets are not included in the image build cache or the final image, using a command like:

Copy

docker build --secret my\_key=my\_value ,src=path/to/my\_secret\_file .

For secrets needed in a running container, **Docker Compose and Kubernetes** offer robust solutions. Docker Compose utilizes a secrets key in the service definition for specifying secret files, as shown in a docker-compose.yml example:

Copy

version: "3.7"

services:

my\_service:

image: centos:7

entrypoint: "cat /run/secrets/my\_secret"

secrets:

- my\_secret

secrets:

my\_secret:

file: ./my\_secret\_file.txt

This configuration allows for the use of secrets when starting services with Docker Compose.

In Kubernetes environments, secrets are natively supported and can be further managed with tools like [Helm-Secrets](https://github.com/futuresimple/helm-secrets). Kubernetes' Role Based Access Controls (RBAC) enhances secret management security, similar to Docker Enterprise.

### gVisor

**gVisor** is an application kernel, written in Go, that implements a substantial portion of the Linux system surface. It includes an [Open Container Initiative (OCI)](https://www.opencontainers.org/) runtime called runsc that provides an **isolation boundary between the application and the host kernel**. The runsc runtime integrates with Docker and Kubernetes, making it simple to run sandboxed containers.

[GitHub - google/gvisor: Application Kernel for ContainersGitHub](https://github.com/google/gvisor)

### Kata Containers

**Kata Containers** is an open source community working to build a secure container runtime with lightweight virtual machines that feel and perform like containers, but provide **stronger workload isolation using hardware virtualization** technology as a second layer of defense.

[Kata Containers - Open Source Container Runtime Softwarekatacontainers](https://katacontainers.io/)

### Summary Tips

* **Do not use the --privileged flag or mount a** [**Docker socket inside the container**](https://raesene.github.io/blog/2016/03/06/The-Dangers-Of-Docker.sock/)**.** The docker socket allows for spawning containers, so it is an easy way to take full control of the host, for example, by running another container with the --privileged flag.
* Do **not run as root inside the container. Use a** [**different user**](https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#user) **and** [**user namespaces**](https://docs.docker.com/engine/security/userns-remap/)**.** The root in the container is the same as on host unless remapped with user namespaces. It is only lightly restricted by, primarily, Linux namespaces, capabilities, and cgroups.
* [**Drop all capabilities**](https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities) **(--cap-drop=all) and enable only those that are required** (--cap-add=...). Many of workloads don’t need any capabilities and adding them increases the scope of a potential attack.
* [**Use the “no-new-privileges” security option**](https://raesene.github.io/blog/2019/06/01/docker-capabilities-and-no-new-privs/) to prevent processes from gaining more privileges, for example through suid binaries.
* [**Limit resources available to the container**](https://docs.docker.com/engine/reference/run/#runtime-constraints-on-resources)**.** Resource limits can protect the machine from denial of service attacks.
* **Adjust** [**seccomp**](https://docs.docker.com/engine/security/seccomp/)**,** [**AppArmor**](https://docs.docker.com/engine/security/apparmor/) **(or SELinux)** profiles to restrict the actions and syscalls available for the container to the minimum required.
* **Use** [**official docker images**](https://docs.docker.com/docker-hub/official_images/) **and require signatures** or build your own based on them. Don’t inherit or use [backdoored](https://arstechnica.com/information-technology/2018/06/backdoored-images-downloaded-5-million-times-finally-removed-from-docker-hub/) images. Also store root keys, passphrase in a safe place. Docker has plans to manage keys with UCP.
* **Regularly** **rebuild** your images to **apply security patches to the host an images.**
* Manage your **secrets wisely** so it's difficult to the attacker to access them.
* If you **exposes the docker daemon use HTTPS** with client & server authentication.
* In your Dockerfile, **favor COPY instead of ADD**. ADD automatically extracts zipped files and can copy files from URLs. COPY doesn’t have these capabilities. Whenever possible, avoid using ADD so you aren’t susceptible to attacks through remote URLs and Zip files.
* Have **separate containers for each micro-s**ervice
* **Don’t put ssh** inside container, “docker exec” can be used to ssh to Container.
* Have **smaller** container **images**

## Docker Breakout / Privilege Escalation

If you are **inside a docker container** or you have access to a user in the **docker group**, you could try to **escape and escalate privileges**:

[PAGEDocker Breakout / Privilege Escalation](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security/docker-breakout-privilege-escalation)

## Docker Authentication Plugin Bypass

If you have access to the docker socket or have access to a user in the **docker group but your actions are being limited by a docker auth plugin**, check if you can **bypass it:**

[PAGEAuthZ& AuthN - Docker Access Authorization Plugin](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security/authz-and-authn-docker-access-authorization-plugin)

## Hardening Docker

* The tool [**docker-bench-security**](https://github.com/docker/docker-bench-security) is a script that checks for dozens of common best-practices around deploying Docker containers in production. The tests are all automated, and are based on the [CIS Docker Benchmark v1.3.1](https://www.cisecurity.org/benchmark/docker/). You need to run the tool from the host running docker or from a container with enough privileges. Find out **how to run it in the README:** [**https://github.com/docker/docker-bench-security**](https://github.com/docker/docker-bench-security).

## References

* <https://blog.trailofbits.com/2019/07/19/understanding-docker-container-escapes/>
* <https://twitter.com/_fel1x/status/1151487051986087936>
* <https://ajxchapman.github.io/containers/2020/11/19/privileged-container-escape.html>
* <https://sreeninet.wordpress.com/2016/03/06/docker-security-part-1overview/>
* <https://sreeninet.wordpress.com/2016/03/06/docker-security-part-2docker-engine/>
* <https://sreeninet.wordpress.com/2016/03/06/docker-security-part-3engine-access/>
* <https://sreeninet.wordpress.com/2016/03/06/docker-security-part-4container-image/>
* <https://en.wikipedia.org/wiki/Linux_namespaces>
* <https://towardsdatascience.com/top-20-docker-security-tips-81c41dd06f57>
* <https://www.redhat.com/sysadmin/privileged-flag-container-engines>
* <https://docs.docker.com/engine/extend/plugins_authorization>
* <https://towardsdatascience.com/top-20-docker-security-tips-81c41dd06f57>
* <https://resources.experfy.com/bigdata-cloud/top-20-docker-security-tips/>

Use [**Trickest**](https://trickest.com/?utm_source=hacktricks&utm_medium=text&utm_campaign=ppc&utm_content=docker-security) to easily build and **automate workflows** powered by the world's **most advanced** community tools. Get Access Today:

[Automate OffSec, EASM, and Custom Security Processes | Trickest](https://trickest.com/?utm_source=hacktricks&utm_medium=banner&utm_campaign=ppc&utm_content=docker-security)

**Learn AWS hacking from zero to hero with** [**htARTE (HackTricks AWS Red Team Expert)**](https://training.hacktricks.xyz/courses/arte)**!**

[PreviousD-Bus Enumeration & Command Injection Privilege Escalation](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/d-bus-enumeration-and-command-injection-privilege-escalation)[NextAbusing Docker Socket for Privilege Escalation](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security/abusing-docker-socket-for-privilege-escalation)

Last updated 2 months ago

* [Basic Docker Engine Security](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#basic-docker-engine-security)
* [Secure Access to Docker Engine](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#secure-access-to-docker-engine)
* [Security of Container Images](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#security-of-container-images)
* [Image Scanning](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#image-scanning)
* [Docker Image Signing](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#docker-image-signing)
* [Containers Security Features](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#containers-security-features)
* [Namespaces](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#namespaces)
* [cgroups](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#cgroups)
* [Capabilities](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#capabilities)
* [Seccomp in Docker](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#seccomp-in-docker)
* [AppArmor in Docker](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#apparmor-in-docker)
* [SELinux in Docker](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#selinux-in-docker)
* [AuthZ & AuthN](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#authz-and-authn)
* [DoS from a container](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#dos-from-a-container)
* [Interesting Docker Flags](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#interesting-docker-flags)
* [--privileged flag](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#privileged-flag)
* [--security-opt](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#security-opt)
* [Other Security Considerations](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#other-security-considerations)
* [Managing Secrets: Best Practices](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#managing-secrets-best-practices)
* [gVisor](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#gvisor)
* [Kata Containers](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#kata-containers)
* [Summary Tips](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#summary-tips)
* [Docker Breakout / Privilege Escalation](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#docker-breakout-privilege-escalation)
* [Docker Authentication Plugin Bypass](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#docker-authentication-plugin-bypass)
* [Hardening Docker](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#hardening-docker)
* [References](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security#references)

Was this helpful?

[Edit on GitHub](https://github.com/HackTricks-wiki/hacktricks/blob/master/linux-hardening/privilege-escalation/docker-security/README.md)

This site uses cookies to deliver its service and to analyse traffic. By browsing this site, you accept the [privacy policy](https://policies.gitbook.com/privacy-and-security/statement/cookies).

AcceptReject