CIS Benchmarks

CIS Docker Benchmark

v1.2.0 - 07-15-2019

Terms of Use

Please see the below link for our current terms of use:

https://www.cisecurity.org/cis-securesuite/cis-securesuite-membership-terms-of-use/

Table of Contents

Overview

This document, CIS Docker 18.09 Benchmark, provides prescriptive guidance for establishing a secure configuration posture for Docker Engine - Community version 18.09 and Docker Enterprise 2.1. This guide was tested against Docker Engine - Community 18.09 on RHEL 7 and Debian 8. It was also tested against Docker Enterprise 2.1, which includes Docker Engine - Enterprise 18.09, Universal Control Plane (UCP) 3.1.0 and Docker Trusted Registry (DTR) 2.6.0, all on RHEL 7 and Debian 8. To obtain the latest version of this guide, please visit http://benchmarks.cisecurity.org. If you have questions, comments, or have identified ways to improve this guide, please write us at feedback@cisecurity.org.

Intended Audience

This document is intended for system and application administrators, security specialists, auditors, help desk, and platform deployment personnel who plan to develop, deploy, assess, or secure solutions that incorporate Docker 18.09 or later and Docker Enterprise 2.1 or later technology on Linux based platforms.

Consensus Guidance

This benchmark was created using a consensus review process comprised of subject matter experts. Consensus participants provide perspective from a diverse set of backgrounds including consulting, software development, audit and compliance, security research, operations, government, and legal.

Each CIS benchmark undergoes two phases of consensus review. The first phase occurs during initial benchmark development. During this phase, subject matter experts convene to discuss, create, and test working drafts of the benchmark. This discussion occurs until consensus has been reached on benchmark recommendations. The second phase begins after the benchmark has been published. During this phase, all feedback provided by the Internet community is reviewed by the consensus team for incorporation in the benchmark. If you are interested in participating in the consensus process, please visit https://workbench.cisecurity.org/.

Typographical Conventions

The following typographical conventions are used throughout this guide:

Convention	Meaning
`Stylized Monospace font`	Used for blocks of code, command, and script examples. Text should be interpreted exactly as presented.
Monospace font	Used for inline code, commands, or examples. Text should be interpreted exactly as presented.
<italic font in brackets>	Italic texts set in angle brackets denote a variable requiring substitution for a real value.
Italic font	Used to denote the title of a book, article, or other publication.
Note	Additional information or caveats

Scoring Information

A scoring status indicates whether compliance with the given recommendation impacts the assessed target's benchmark score. The following scoring statuses are used in this benchmark:

Scored

Failure to comply with "Scored" recommendations will decrease the final benchmark score. Compliance with "Scored" recommendations will increase the final benchmark score.

Not Scored

Failure to comply with "Not Scored" recommendations will not decrease the final benchmark score. Compliance with "Not Scored" recommendations will not increase the final benchmark score.

Profile Definitions

The following configuration profiles are defined by this Benchmark:

- **Level 1 - Docker - Linux**

  Items in this profile intend to:

  - Be practical and prudent;
  - Provide a clear security benefit; and
  - Not inhibit the utility of the technology beyond acceptable means.

- **Level 1 - Linux Host OS**

  Items in this profile pertain to the Linux Host OS and intend to:

  - Be practical and prudent;
  - Provide a clear security benefit; and
  - Not inhibit the utility of the technology beyond acceptable means.

- **Level 1 - Docker Engine - Enterprise**

  Items in this profile pertain to the Docker EE and intend to:

  - Be practical and prudent;
  - Provide a clear security benefit; and
  - Not inhibit the utility of the technology beyond acceptable means.

- **Level 2 - Docker - Linux**

  Items in this profile exhibit one or more of the following characteristics:

  - Are intended for environments or use cases where security is paramount
  - Acts as defense in depth measure
  - May negatively inhibit the utility or performance of the technology

- **Level 2 - Linux Host OS**

  Items in this profile exhibit one or more of the following characteristics:

  - Are intended for environments or use cases where security is paramount
  - Acts as defense in depth measure
  - May negatively inhibit the utility or performance of the technology

- **Level 2 - Docker Engine - Enterprise**

  Items in this profile exhibit one or more of the following characteristics:

  - Are intended for environments or use cases where security is paramount
  - Acts as defense in depth measure
  - May negatively inhibit the utility or performance of the technology

Acknowledgements

This benchmark exemplifies the great things a community of users, vendors, and subject matter experts can accomplish through consensus collaboration. The CIS community thanks the entire consensus team with special recognition to the following individuals who contributed greatly to the creation of this guide:

Editor
Rory Mccune
Thomas Sjögren https://github.com/konstruktoid

Contributor
Brian Andrzejewski
Andrew Weiss $(ISC)^2$ Certified Authorization Professional
Ryan Puffer , Microsoft
Marion McCune

Recommendations

1 Host Configuration

This section covers security recommendations that you should follow to prepare the host machine that you plan to use for executing containerized workloads. Securing the Docker host and following your infrastructure security best practices would build a solid and secure foundation for executing containerized workloads.

1.1 General Configuration

This section contains general host recommendations for systems running Docker

1.1.1 Ensure the container host has been Hardened (Not Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

A container host is able to run one or more containers. It is of utmost importance to harden the host to mitigate host security misconfiguration.

Rationale:

You should follow infrastructure security best practices and harden your host OS. Keeping the host system hardened will ensure that host vulnerabilities are mitigated. Not hardening the host system could lead to security exposures and breaches.

Audit:

Ensure that the host specific security guidelines are followed. Ask the system administrators which security benchmark the current host system should currently be compliant with and check that security standards associated with this standard are currently in place.

Remediation:

You may consider various CIS Security Benchmarks for your container host. If you have other security guidelines or regulatory requirements to adhere to, please follow them as suitable in your environment.

Impact:

None.

Default Value:

By default, the host has factory setting and is not hardened.

References:

1. https://docs.docker.com/engine/security/security/

2. https://learn.cisecurity.org/benchmarks

CIS Controls:

Version 6

3 Secure Configurations for Hardware and Software on Mobile Devices, Laptops, Workstations, and Servers

Secure Configurations for Hardware and Software on Mobile Devices, Laptops, Workstations, and Servers

1.1.2 Ensure that the version of Docker is up to date (Not Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Frequent releases for Docker are issued which address security vulnerabilities, resolve product bugs and bring in new functionality. You should keep a tab on these product updates and upgrade as frequently as possible in line with the general IT security policy of your organization.

Rationale:

By staying up to date on Docker updates, vulnerabilities in the software can be mitigated. An experienced attacker may be able to exploit known vulnerabilities resulting in them being able to attain inappropriate access or to elevate their privileges. If you do not ensure that Docker is running at the most current release consistent with the requirements of of your application, you may introduce unwanted behaviour and it is therefore important to ensure that you monitor software versions and upgrade in a timely fashion.

Audit:

You should execute the command below in order to verify that the Docker version is up to date in line with the requirements of the application you are running. It should be noted that it is not a security requirement to be at the most up to date version, provided the version you are using does not contain any critical or high security vulnerabilities.

```
docker version
```

Remediation:

You should monitor versions of Docker releases and make sure your software is updated as required.

Impact:

You should perform a risk assessment regarding Docker version updates and review how they may impact your operations. You should be aware that third-party products that use Docker may require older major versions of Docker to be supported, and this should be reviewed in line with the general IT security policy of your organization, particularly where security vulnerabilities in older versions have been publicly disclosed.

Default Value:

Not Applicable

References:

1. https://docs.docker.com/engine/installation/
2. https://github.com/moby/moby/releases/latest
3. https://github.com/docker/docker-ce/releases/latest

CIS Controls:

Version 6

4 Continuous Vulnerability Assessment and Remediation
Continuous Vulnerability Assessment and Remediation

1.2 Linux Hosts Specific Configuration

This section contains recommendations that securing Linux Hosts running Docker Containers.

1.2.1 Ensure a separate partition for containers has been created (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

All Docker containers and their data and metadata is stored under `/var/lib/docker` directory. By default, `/var/lib/docker` should be mounted under either the `/` or `/var` partitions dependent on how the Linux operating system in use is configured.

Rationale:

Docker depends on `/var/lib/docker` as the default directory where all Docker related files, including the images, are stored. This directory could fill up quickly causing both Docker and the host to become unusable. For this reason, you should create a separate partition (logical volume) for storing Docker files.

Audit:

At the Docker host execute one of the below commands:

```
grep '/var/lib/docker\s' /proc/mounts
```

This should return the partition details for the `/var/lib/docker` mountpoint.

```
mountpoint -- "$(docker info -f '{{ .DockerRootDir }}')"
```

This should return whether the configured root directory is a mount point.

Remediation:

For new installations, you should create a separate partition for the `/var/lib/docker` mount point. For systems which have already been installed, you should use the Logical Volume Manager (LVM) within Linux to create a new partition.

Impact:

None.

Default Value:

By default, `/var/lib/docker` is mounted under the `/` or `/var` partitions dependent on how the OS is configured.

References:

1. https://www.projectatomic.io/docs/docker-storage-recommendation/

CIS Controls:

Version 6

14 Controlled Access Based on the Need to Know
Controlled Access Based on the Need to Know

1.2.2 Ensure only trusted users are allowed to control Docker daemon (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

The Docker daemon currently requires access to the Docker socket which is, by default, owned by the user `root` and the group `docker`.

Rationale:

Docker allows you to share a directory between the Docker host and a guest container without limiting the access rights of the container. This means that you can start a container and map the `/` directory on your host to the container. The container would then be able to modify your host file system without any restrictions. This means that you could gain elevated privileges simply by being a member of the `docker` group and subsequently start a container which maps the root `/` directory on the host.

Audit:

Execute the following command on the docker host and ensure that only trusted users are members of the `docker` group.

```
getent group docker
```

Remediation:

You should remove any untrusted users from the `docker` group. Additionally, you should not create a mapping of sensitive directories from the host to container volumes.

Impact:

Provided the proceeding instructions are implemented, rights to build and execute containers as normal user would be restricted.

Default Value:

Not Applicable

References:

1. https://docs.docker.com/engine/security/security/#docker-daemon-attack-surface
2. https://www.andreas-jung.com/contents/on-docker-security-docker-group-considered-harmful
3. http://www.projectatomic.io/blog/2015/08/why-we-dont-let-non-root-users-run-docker-in-centos-fedora-or-rhel/

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

1.2.3 Ensure auditing is configured for the Docker daemon (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit all Docker daemon activities.

Rationale:

As well as auditing the normal Linux file system and system calls, you should also audit the Docker daemon. Because this daemon runs with `root` privileges. It is very important to audit its activities and usage.

Audit:

Verify that there are audit rules for the Docker daemon. For example, you could execute the following command:

```
auditctl -l | grep /usr/bin/dockerd
```

This should show the rules associated with the Docker daemon.

Remediation:

You should add rules for the Docker daemon.
For example:
Add the line below to the `/etc/audit/audit.rules` file:

```
-w /usr/bin/dockerd -k docker
```

Then, restart the audit daemon using the following command

```
service auditd restart
```

Impact:

Auditing can generate large log files. You should ensure that these are rotated and archived periodically. A separate partition should also be created for audit logs to avoid filling up any other critical partition.

Default Value:

By default, the Docker daemon is not audited.

References:

1. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html

CIS Controls:

Version 6

6.2 Ensure Audit Log Settings Support Appropriate Log Entry Formatting

Validate audit log settings for each hardware device and the software installed on it, ensuring that logs include a date, timestamp, source addresses, destination addresses, and various other useful elements of each packet and/or transaction. Systems should record logs in a standardized format such as syslog entries or those outlined by the Common Event Expression initiative. If systems cannot generate logs in a standardized format, log normalization tools can be deployed to convert logs into such a format.

Version 7

6.2 Activate audit logging

Ensure that local logging has been enabled on all systems and networking devices.

6.3 Enable Detailed Logging

Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.

1.2.4 Ensure auditing is configured for Docker files and directories - /var/lib/docker (Scored)

Profile Applicability:

- Level 2 - Linux Host OS

Description:

Audit `/var/lib/docker`.

Rationale:

As well as auditing the normal Linux file system and system calls, you should also audit all Docker related files and directories. The Docker daemon runs with `root` privileges and its behaviour depends on some key files and directories. `/var/lib/docker` is one such directory. As it holds all the information about containers it should be audited.

Audit:

You should verify that there is an audit rule applied to the `/var/lib/docker` directory.
For example, you could execute the command below:

```
auditctl -l | grep /var/lib/docker
```

This should list a rule for the `/var/lib/docker` directory.

Remediation:

You should add a rule for the `/var/lib/docker` directory.
For example,
Add the line as below to the `/etc/audit/audit.rules` file:

```
-w /var/lib/docker -k docker
```

Then restart the audit daemon.
For example,

```
service auditd restart
```

Impact:

Auditing can generate large log files. You should ensure that these are rotated and archived periodically. A separate partition should also be created for audit logs to avoid filling up any other critical partition.

Default Value:

By default, Docker related files and directories are not audited.

References:

1. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html

CIS Controls:

Version 6

14.6 Enforce Detailed Audit Logging For Sensitive Information
Enforce detailed audit logging for access to nonpublic data and special authentication for sensitive data.

Version 7

14.9 Enforce Detail Logging for Access or Changes to Sensitive Data
Enforce detailed audit logging for access to sensitive data or changes to sensitive data (utilizing tools such as File Integrity Monitoring or Security Information and Event Monitoring).

1.2.5 Ensure auditing is configured for Docker files and directories - /etc/docker (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `/etc/docker`.

Rationale:

As well as auditing the normal Linux file system and system calls, you should also audit all Docker related files and directories. The Docker daemon runs with `root` privilege and its behavior depends on some key files and directories, one of these being `/etc/docker`. This holds various certificates and keys used for TLS communication between Docker daemon and Docker client and as such it should be audited.

Audit:

You should verify that there is an audit rule applied to the `/etc/docker` directory.
For example, you could execute the command below:

```
auditctl -l | grep /etc/docker
```

This should display a rule for the `/etc/docker` directory.

Remediation:

You should add a rule for the `/etc/docker` directory.
For example:
Add the line below to the `/etc/audit/audit.rules` file:

```
-w /etc/docker -k docker
```

Then restart the audit daemon. For example:

```
service auditd restart
```

Impact:

Auditing can generate large log files. You should ensure that these are rotated and archived periodically. A separate partition should also be created for audit logs to avoid filling up any other critical partition.

Default Value:

By default, Docker related files and directories are not audited.

References:

1. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html

CIS Controls:

Version 6

14.6 Enforce Detailed Audit Logging For Sensitive Information
Enforce detailed audit logging for access to nonpublic data and special authentication for sensitive data.

Version 7

14.9 Enforce Detail Logging for Access or Changes to Sensitive Data
Enforce detailed audit logging for access to sensitive data or changes to sensitive data (utilizing tools such as File Integrity Monitoring or Security Information and Event Monitoring).

1.2.6 Ensure auditing is configured for Docker files and directories - docker.service (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit the `docker.service` if applicable.

Rationale:

As well as auditing the normal Linux file system and system calls, you should also audit all Docker related files and directories. The Docker daemon runs with `root` privileges and its behavior depends on some key files and directories with `docker.service` being one such file. The `docker.service` file might be present if the daemon parameters have been changed by an administrator. If so, it holds various parameters for the Docker daemon and should be audited.

Audit:

Step 1: Find out the file location:

```
systemctl show -p FragmentPath docker.service
```

Step 2: If the file does not exist, this recommendation does not apply. If the file does exist, verify that there is an audit rule corresponding to the file:
For example, you could execute the command below:

```
auditctl -l | grep docker.service
```

This should display a rule for `docker.service`.

Remediation:

If the file exists, a rule for it should be added.
For example:
Add the line as below in `/etc/audit/audit.rules` file:

```
-w /usr/lib/systemd/system/docker.service -k docker
```

Then restart the audit daemon.
For example:

```
service auditd restart
```

Impact:

Auditing can generate large log files. You should ensure that these are rotated and archived periodically. A separate partition should also be created for audit logs to avoid filling up any other critical partition.

Default Value:

By default, Docker related files and directories are not audited. The file `docker.service` may not be present on the system.

References:

1. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html

CIS Controls:

Version 6

14.6 Enforce Detailed Audit Logging For Sensitive Information
Enforce detailed audit logging for access to nonpublic data and special authentication for sensitive data.

Version 7

14.9 Enforce Detail Logging for Access or Changes to Sensitive Data
Enforce detailed audit logging for access to sensitive data or changes to sensitive data (utilizing tools such as File Integrity Monitoring or Security Information and Event Monitoring).

1.2.7 Ensure auditing is configured for Docker files and directories - docker.socket (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `docker.socket`, if applicable.

Rationale:

As well as auditing the normal Linux file system and system calls, you should also audit the Docker daemon. Because this daemon runs with root privileges, it is very important to audit its activities and usage. Its behavior depends on some key files and directories with `docker.socket` being one such file, and as this holds various parameters for the Docker daemon, it should be audited.

Audit:

Step 1: Find out the file location:

```
systemctl show -p FragmentPath docker.socket
```

Step 2: If the file does not exist, this recommendation is not applicable. If the file exists, you should verify that there is an audit rule corresponding to the file:
For example, you could execute the command below:

```
auditctl -l | grep docker.socket
```

This should display a rule for `docker.socket`.

Remediation:

If the file exists, you should add a rule for it.
For example:
Add the line below to the `/etc/audit/audit.rules` file:

```
-w /usr/lib/systemd/system/docker.socket -k docker
```

Then restart the audit daemon.
For example:

```
service auditd restart
```

Impact:

Auditing can generate large log files. You should ensure that these are rotated and archived periodically. A separate partition should also be created for audit logs to avoid filling up any other critical partition.

Default Value:

By default, Docker related files and directories are not audited. The file `docker.socket` may not be present, but if it is, it should be audited.

References:

1. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html

CIS Controls:

Version 6

14.6 Enforce Detailed Audit Logging For Sensitive Information
Enforce detailed audit logging for access to nonpublic data and special authentication for sensitive data.

Version 7

14.9 Enforce Detail Logging for Access or Changes to Sensitive Data
Enforce detailed audit logging for access to sensitive data or changes to sensitive data (utilizing tools such as File Integrity Monitoring or Security Information and Event Monitoring).

1.2.8 Ensure auditing is configured for Docker files and directories - /etc/default/docker (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `/etc/default/docker`, if applicable.

Rationale:

As well as auditing the normal Linux file system and system calls, you should audit all Docker related files and directories. The Docker daemon runs with `root` privileges and its behavior depends on some key files and directories. `/etc/default/docker` is one such file. It holds various parameters related to the Docker daemon and should therefore be audited.

Audit:

You should verify that there is an audit rule associated with the `/etc/default/docker` file. For example, you could execute the command below:

```
auditctl -l | grep /etc/default/docker
```

This should display a rule for the `/etc/default/docker` file.

Remediation:

You should add a rule for the `/etc/default/docker` file.
For example:
Add the line below to the `/etc/audit/audit.rules` file:

```
-w /etc/default/docker -k docker
```

Then restart the audit daemon.
For example:

```
service auditd restart
```

Impact:

Auditing can generate large log files. You should ensure that these are rotated and archived periodically. A separate partition should also be created for audit logs to avoid filling up any other critical partition.

Default Value:

By default, Docker related files and directories are not audited so these defaults should be changed in line with organizational security policy. The file `/etc/default/docker` may not be present, and if so, this recommendation is not applicable.

References:

1. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html

CIS Controls:

Version 6

14.6 Enforce Detailed Audit Logging For Sensitive Information
Enforce detailed audit logging for access to nonpublic data and special authentication for sensitive data.

Version 7

14.9 Enforce Detail Logging for Access or Changes to Sensitive Data
Enforce detailed audit logging for access to sensitive data or changes to sensitive data (utilizing tools such as File Integrity Monitoring or Security Information and Event Monitoring).

1.2.9 Ensure auditing is configured for Docker files and directories - /etc/sysconfig/docker (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `/etc/sysconfig/docker` if applicable

Rationale:

As well as auditing the normal Linux file system and system calls, you should also audit the Docker daemon. Because this daemon runs with root privileges it is very important to audit its activities and usage. Its behavior depends on some key files and directories and `/etc/sysconfig/docker` is one such file as it contains various parameters related to the Docker daemon when run on CentOS and RHEL based distributions. If present, it is important that it is audited.

Audit:

You should verify that there is an audit rule present relating to the `/etc/sysconfig/docker` file.
For example, you could execute the command below:

```
auditctl -l | grep /etc/sysconfig/docker
```

This should display a rule for `/etc/sysconfig/docker` file.

Remediation:

You should add a rule for `/etc/sysconfig/docker` file.
For example:
Add the line below to the `/etc/audit/audit.rules` file:

```
-w /etc/sysconfig/docker -k docker
```

Then restart the audit daemon.
For example:

```
service auditd restart
```

Impact:

Auditing can generate large log files. You should ensure that these are rotated and archived periodically. A separate partition should also be created for audit logs to avoid filling up any other critical partition.

Default Value:

By default, Docker related files and directories are not audited. The file `/etc/sysconfig/docker` may not be presemt on the system and in that case, this recommendation is not applicable.

References:

1. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html

CIS Controls:

Version 6

14.6 Enforce Detailed Audit Logging For Sensitive Information
Enforce detailed audit logging for access to nonpublic data and special authentication for sensitive data.

Version 7

14.9 Enforce Detail Logging for Access or Changes to Sensitive Data
Enforce detailed audit logging for access to sensitive data or changes to sensitive data (utilizing tools such as File Integrity Monitoring or Security Information and Event Monitoring).

1.2.10 Ensure auditing is configured for Docker files and directories - /etc/docker/daemon.json (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `/etc/docker/daemon.json`, if applicable.

Rationale:

As well as auditing the normal Linux file system and system calls, you should also audit all Docker related files and directories. The Docker daemon runs with `root` privileges and its behavior depends on some key files and directories. `/etc/docker/daemon.json` is one such file. This holds various parameters for the Docker daemon, and as such it should be audited.

Audit:

You should verify that there is an audit rule associated with the `/etc/docker/daemon.json` file.
For example, you could execute the command below:

```
auditctl -l | grep /etc/docker/daemon.json
```

This should display a rule for the `/etc/docker/daemon.json` file.

Remediation:

You should add a rule for the `/etc/docker/daemon.json` file.
For example:
Add the line below to the `/etc/audit/audit.rules` file:

```
-w /etc/docker/daemon.json -k docker
```

Then restart the audit daemon.
For example:

```
service auditd restart
```

Impact:

Auditing can generate large log files. You should ensure that these are rotated and archived periodically. A separate partition should also be created for audit logs to avoid filling up any other critical partition.

Default Value:

By default, Docker related files and directories are not audited. The file `/etc/docker/daemon.json` may not exist on the system and in that case, this recommendation is not applicable.

References:

1. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html
2. https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file

CIS Controls:

Version 6

14.6 Enforce Detailed Audit Logging For Sensitive Information
Enforce detailed audit logging for access to nonpublic data and special authentication for sensitive data.

Version 7

14.9 Enforce Detail Logging for Access or Changes to Sensitive Data
Enforce detailed audit logging for access to sensitive data or changes to sensitive data (utilizing tools such as File Integrity Monitoring or Security Information and Event Monitoring).

1.2.11 Ensure auditing is configured for Docker files and directories - /usr/bin/containerd (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `/usr/bin/containerd` if applicable.

Rationale:

As well as auditing the normal Linux file system and system calls, you should audit all Docker related files and directories. The Docker daemon runs with `root` privileges and its behavior depends on some key files and directories. `/usr/bin/containerd` is one such file and as such should be audited.

Audit:

You should verify that there is an audit rule corresponding to `/usr/bin/containerd` file.
For example, you could execute the command below:

```
auditctl -l | grep /usr/bin/containerd
```

This should display a rule for `/usr/bin/containerd` file.

Remediation:

You should add a rule for the `/usr/bin/containerd` file.
For example:
Add the line below to the `/etc/audit/audit.rules` file:

```
-w /usr/bin/containerd -k docker
```

Then restart the audit daemon.
For example:

```
service auditd restart
```

Impact:

Auditing can generate large log files. You should ensure that these are rotated and archived periodically. A separate partition should also be created for audit logs to avoid filling up any other critical partition.

Default Value:

By default, Docker related files and directories are not audited. The file `/usr/bin/containerd` may not be present on the system and in that case, this recommendation is not applicable.

References:

1. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html
2. https://github.com/docker/docker/pull/20662
3. https://containerd.tools/

CIS Controls:

Version 6

14.6 Enforce Detailed Audit Logging For Sensitive Information
Enforce detailed audit logging for access to nonpublic data and special authentication for sensitive data.

Version 7

14.9 Enforce Detail Logging for Access or Changes to Sensitive Data
Enforce detailed audit logging for access to sensitive data or changes to sensitive data (utilizing tools such as File Integrity Monitoring or Security Information and Event Monitoring).

1.2.12 Ensure auditing is configured for Docker files and directories - /usr/sbin/runc (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `/usr/sbin/runc` if applicable

Rationale:

As well as auditing the normal Linux file system and system calls, you should also audit all Docker related files and directories. The Docker daemon runs with `root` privileges and its behavior depends on some key files and directories. `/usr/sbin/runc` is one such file, and as such it should be audited.

Audit:

You should verify that there is an audit rule corresponding to `/usr/sbin/runc` file.
For example, you could execute the command below:

```
auditctl -l | grep /usr/sbin/runc
```

This should display a rule for the `/usr/sbin/runc` file.

Remediation:

You should add a rule for `/usr/sbin/runc` file.
For example:
Add the line below to the `/etc/audit/audit.rules` file:

```
-w /usr/sbin/runc -k docker
```

Then restart the audit daemon.
For example:

```
service auditd restart
```

Impact:

Auditing can generate large log files. You should ensure that these are rotated and archived periodically. A separate partition should also be created for audit logs to avoid filling up any other critical partition.

Default Value:

By default, Docker related files and directories are not audited. The file `/usr/sbin/runc` may not be present on the system and in that case this recommendation is not applicable.

References:

1. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html
2. https://github.com/docker/docker/pull/20662
3. https://containerd.tools/
4. https://github.com/opencontainers/runc

CIS Controls:

Version 6

14.6 Enforce Detailed Audit Logging For Sensitive Information
Enforce detailed audit logging for access to nonpublic data and special authentication for sensitive data.

Version 7

14.9 Enforce Detail Logging for Access or Changes to Sensitive Data
Enforce detailed audit logging for access to sensitive data or changes to sensitive data (utilizing tools such as File Integrity Monitoring or Security Information and Event Monitoring).

2 Docker daemon configuration

This section lists the recommendations that alter and secure the behavior of the Docker daemon. The settings that are under this section affect ALL container instances.

Note: Docker daemon options can also be controlled using files such as `/etc/sysconfig/docker`, `/etc/default/docker`, the `systemd` unit file or `/etc/docker/daemon.json`. Also, note that Docker in daemon mode can be identified as `/usr/bin/dockerd`, or having `-d` or `daemon` as the argument to `docker` service.

2.1 Ensure network traffic is restricted between containers on the default bridge (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

By default, all network traffic is allowed between containers on the same host on the default network bridge. If not desired, restrict all inter-container communication. Link specific containers together that require communication. Alternatively, you can create custom network and only join containers that need to communicate to that custom network.

Rationale:

By default, unrestricted network traffic is enabled between all containers on the same host on the default network bridge. Thus, each container has the potential of reading all packets across the container network on the same host. This might lead to an unintended and unwanted disclosure of information to other containers. Hence, restrict inter-container communication on the default network bridge.

Audit:

Run the below command and verify that the default network bridge has been configured to restrict inter-container communication.

```
docker network ls --quiet | xargs docker network inspect --format '{{ .Name
}}: {{ .Options }}'
```

It should return `com.docker.network.bridge.enable_icc:false` for the default network bridge.

Remediation:

Edit the Docker daemon configuration file to ensure that icc is disabled. It should include the following setting

```
"icc": false
```

Alernatively, run the docker daemon directly and pass `--icc=false` as an argument.
For Example,

```
dockerd --icc=false
```

Alternatively, you can follow the Docker documentation and create a custom network and only join containers that need to communicate to that custom network. The `--icc` parameter only applies to the default docker bridge, if custom networks are used then the approach of segmenting networks should be adopted instead.

Impact:

Inter-container communication would be disabled on the default network bridge. If any communication between containers on the same host is desired, then it needs to be explicitly defined using container linking or alternatively custom networks have to be defined.

Default Value:

By default, all inter-container communication is allowed on the default network bridge.

References:

1. https://docs.docker.com/engine/userguide/networking/
2. https://docs.docker.com/engine/userguide/networking/default_network/container-communication/#communication-between-containers

2.2 Ensure the logging level is set to 'info' (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

Set Docker daemon log level to `info`.

Rationale:

Setting up an appropriate log level, configures the Docker daemon to log events that you would want to review later. A base log level of `info` and above would capture all logs except debug logs. Until and unless required, you should not run Docker daemon at `debug` log level.

Audit:

To confirm this setting a combination of reviewing the dockerd start-up options and a review of any settings in `/etc/docker/daemon.json` should be completed.
To review the dockerd startup options, use:

```
ps -ef | grep dockerd
```

Ensure that either the `--log-level` parameter is not present or if present, then it is set to `info`.
The contents of `/etc/docker/daemon.json` should also be reviewed for this setting.

Remediation:

Ensure that the Docker daemon configuration file has the following configuration included

```
"log-level": "info"
```

Alternatively, run the Docker daemon as below:

```
dockerd --log-level="info"
```

Impact:

None.

Default Value:

By default, Docker daemon is set to log level of `info`.

References:

1. https://docs.docker.com/edge/engine/reference/commandline/dockerd/

CIS Controls:

Version 6

6.2 Ensure Audit Log Settings Support Appropriate Log Entry Formatting

Validate audit log settings for each hardware device and the software installed on it, ensuring that logs include a date, timestamp, source addresses, destination addresses, and various other useful elements of each packet and/or transaction. Systems should record logs in a standardized format such as syslog entries or those outlined by the Common Event Expression initiative. If systems cannot generate logs in a standardized format, log normalization tools can be deployed to convert logs into such a format.

Version 7

6.2 Activate audit logging

Ensure that local logging has been enabled on all systems and networking devices.

6.3 Enable Detailed Logging

Enable system logging to include detailed information such as an event source, date, user, timestamp, source addresses, destination addresses, and other useful elements.

2.3 Ensure Docker is allowed to make changes to iptables (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

The iptables firewall is used to set up, maintain, and inspect the tables of IP packet filter rules within the Linux kernel. The Docker daemon should be allowed to make changes to the `iptables` ruleset.

Rationale:

Docker will never make changes to your system `iptables` rules unless you allow it to do so. If you do allow this, Docker server will automatically make any required changes. We recommended letting Docker make changes to `iptables`automatically in order to avoid networking misconfigurations that could affect the communication between containers and with the outside world. Additionally, this reduces the administrative overhead of updating `iptables`every time you add containers or modify networking options.

Audit:

To confirm this setting you should review the dockerd start-up options and the settings in `/etc/docker/daemon.json`
To review the dockerd startup options, use:

```
ps -ef | grep dockerd
```

Ensure that the `--iptables` parameter is either not present or not set to `false`.
The contents of `/etc/docker/daemon.json` should also be reviewed for this setting.

Remediation:

Do not run the Docker daemon with `--iptables=false` parameter. For example, do not start the Docker daemon as below:

```
dockerd --iptables=false
```

Impact:

The Docker daemon service requires iptables rules to be enabled before it starts. Any restarts of iptables during Docker daemon operation may result in losing Docker created

rules. Adding iptables-persistent to your iptables install can assist with mitigation of this impact.

Default Value:

By default, `iptables` is set to `true`.

References:

1. https://docs.docker.com/engine/userguide/networking/default_network/container-communication/
2. https://fralef.me/docker-and-iptables.html

CIS Controls:

Version 6

5 Controlled Use of Administration Privileges
Controlled Use of Administration Privileges

2.4 Ensure insecure registries are not used (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

Docker considers a private registry either secure or insecure. By default, registries are considered secure.

Rationale:

A secure registry uses TLS. A copy of registry's CA certificate is placed on the Docker host at `/etc/docker/certs.d/<registry-name>/` directory. An insecure registry is one which does not have a valid registry certificate, or one not not using TLS. Insecure registries should not be used as they present a risk of traffic interception and modification.

Additionally, once a registry has been marked as insecure commands such as `docker pull`, `docker push`, and `docker search` will not result in an error message and users may indefinitely be working with this type of insecure registry without ever being notified of the risk of potential compromise.

Audit:

You should execute the command below to find out if any insecure registries are in use:

```
docker info --format 'Insecure Registries:
{{.RegistryConfig.InsecureRegistryCIDRs}}'
```

Remediation:

You should ensure that no insecure registries are in use.

Impact:

None.

Default Value:

By default, Docker assumes all, but local, registries are secure.

References:

1. https://docs.docker.com/registry/insecure/

CIS Controls:

Version 6

14.2 Encrypt All Sensitive Information Over Less-trusted Networks

All communication of sensitive information over less-trusted networks should be encrypted. Whenever information flows over a network with a lower trust level, the information should be encrypted.

Version 7

14.4 Encrypt All Sensitive Information in Transit

Encrypt all sensitive information in transit.

2.5 Ensure aufs storage driver is not used (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

Do not use `aufs` as the storage driver for your Docker instance.

Rationale:

The `aufs` storage driver is the oldest storage driver used on Linux systems. It is based on a Linux kernel patch-set that is unlikely in future to be merged into the main OS kernel. The `aufs` driver is also known to cause some serious kernel crashes. `aufs` only has legacy support within systems using Docker. Most importantly, `aufs` is not a supported driver in many Linux distributions using latest Linux kernels.

Audit:

Execute the below command and verify that `aufs` is not used as storage driver:

```
docker info --format 'Storage Driver: {{ .Driver }}'
```

The above command should not return `aufs`.

Remediation:

Do not explicitly use `aufs` as storage driver.
For example, do not start Docker daemon as below:

```
dockerd --storage-driver aufs
```

Impact:

`aufs` is the only storage driver that allows containers to share executable and shared library memory. It might be useful if you are running thousands of containers with the same program or libraries, however its use should be reviewed in line with your organization's security policy.

Default Value:

By default, Docker uses `devicemapper` as the storage driver on most of the platforms. The default storage driver can vary based on your OS vendor. You should use the storage driver

that is recommended by your preferred vendor and which is in line with policy around the applications which are being deployed.

References:

1. https://docs.docker.com/engine/userguide/storagedriver/selectadriver/#supported-backing-filesystems
2. http://muehe.org/posts/switching-docker-from-aufs-to-devicemapper/
3. http://jpetazzo.github.io/assets/2015-03-05-deep-dive-into-docker-storage-drivers.html#1
4. https://docs.docker.com/engine/userguide/storagedriver/

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

2.6 Ensure TLS authentication for Docker daemon is configured (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

It is possible to make the Docker daemon available remotely over a TCP port. If this is required, you should ensure that TLS authentication is configured in order to restrict access to the Docker daemon via IP address and port.

Rationale:

By default, the Docker daemon binds to a non-networked Unix socket and runs with `root` privileges. If you change the default Docker daemon binding to a TCP port or any other Unix socket, anyone with access to that port or socket could have full access to the Docker daemon and therefore in turn to the host system. For this reason, you should not bind the Docker daemon to another IP/port or a Unix socket.

If you must expose the Docker daemon via a network socket, you should configure TLS authentication for the daemon and for any Docker Swarm APIs (if they are in use). This type of configuration restricts the connections to your Docker daemon over the network to a limited number of clients who have access to the TLS client credentials.

Audit:

To confirm this setting, review the dockerd start-up options and any settings in `/etc/docker/daemon.json`.
To review the dockerd startup options, use:

```
ps -ef | grep dockerd
```

Ensure that the below parameters are present:

```
--tlsverify
--tlscacert
--tlscert
--tlskey
```

The contents of `/etc/docker/daemon.json` to ensure these settings are in place.

Remediation:

Follow the steps mentioned in the Docker documentation or other references.

Impact:

You would need to manage and guard certificates and keys for the Docker daemon and Docker clients.

Default Value:

By default, TLS authentication is not configured.

References:

1. https://docs.docker.com/engine/security/https/

CIS Controls:

Version 6

9.1 Limit Open Ports, Protocols, and Services
Ensure that only ports, protocols, and services with validated business needs are running on each system.

Version 7

9.2 Ensure Only Approved Ports, Protocols and Services Are Running
Ensure that only network ports, protocols, and services listening on a system with validated business needs, are running on each system.

2.7 Ensure the default ulimit is configured appropriately (Not Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

Set the default ulimit options as appropriate in your environment.

Rationale:

`ulimit` provides control over the resources available to the shell and to processes which it starts. Setting system resource limits judiciously can save you from disasters such as a fork bomb. On occasion, even friendly users and legitimate processes can overuse system resources and can make the system unusable.

Setting the default ulimit for the Docker daemon enforces the ulimit for all container instances. In this case you would not need to setup ulimit for each container instance. However, the default ulimit can be overridden during container runtime, if needed. Therefore, in order to have proper control over system resources, define a default ulimit as is needed in your environment.

Audit:

To confirm this setting you should review the dockerd start-up options and any settings in `/etc/docker/daemon.json`.
To review the dockerd startup options, use:

```
ps -ef | grep dockerd
```

Ensure that the `--default-ulimit` parameter is set as appropriate.
The contents of `/etc/docker/daemon.json` should also be reviewed for this setting.

Remediation:

Run Docker in daemon mode and pass `--default-ulimit` as argument with respective ulimits as appropriate in your environment and in line with your security policy.
For Example,

```
dockerd --default-ulimit nproc=1024:2048 --default-ulimit nofile=100:200
```

Impact:

If ulimits are set incorrectly this could cause issues with system resources, possibly causing a denial of service condition.

Default Value:

By default, no ulimit is set.

References:

1. https://docs.docker.com/edge/engine/reference/commandline/dockerd/#default-ulimits

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

2.8 Enable user namespace support (Scored)

Profile Applicability:

- Level 2 - Docker - Linux

Description:

You should enable user namespace support in Docker daemon to utilize container user to host user re-mapping. This recommendation is beneficial where the containers you are using do not have an explicit container user defined in the container image. If the container images that you are using have a pre-defined non-root user, this recommendation may be skipped as this feature is still in its infancy, and might result in unpredictable issues or difficulty in configuration.

Rationale:

The Linux kernel "user namespace" support within the Docker daemon provides additional security for the Docker host system. It allows a container to have a unique range of user and group IDs which are outside the traditional user and group range utilized by the host system.

For example, the root user can have the expected administrative privileges inside the container but can effectively be mapped to an unprivileged UID on the host system.

Audit:

```
ps -p $(docker inspect --format='{{ .State.Pid }}' <CONTAINER ID>) -o
pid,user
```

The above command will find the PID of the container and then list the host user associated with the container process. If the container process is running as `root`, then this configuration may be non-compliant with your organization's security policy..
Alternatively, you can run `docker info` to ensure that the `userns` is listed under `Security Options:`

```
docker info --format '{{ .SecurityOptions }}'
```

Remediation:

Please consult the Docker documentation for various ways in which this can be configured depending upon your requirements. Your steps might also vary based on platform - For example, on Red Hat, sub-UIDs and sub-GIDs mapping creation do not work automatically. You might have to create your own mapping.

The high-level steps are as below:
Step 1: Ensure that the files `/etc/subuid` and `/etc/subgid` exist.

```
touch /etc/subuid /etc/subgid
```

Step 2: Start the docker daemon with `--userns-remap` flag

```
dockerd --userns-remap=default
```

Impact:

User namespace remapping is incompatible with a number of Docker features and also currently breaks some of its functionalities. Reference the Docker documentation and included links for details.

Default Value:

By default, user namespace is not remapped. Consideration should be given to implementing this in line with the requirements of the applications being used and the organization's security policy.

References:

1. http://man7.org/linux/man-pages/man7/user_namespaces.7.html
2. https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-user-namespace-options
3. http://events.linuxfoundation.org/sites/events/files/slides/User%20Namespaces%20-%20ContainerCon%202015%20-%2016-9-final_0.pdf

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

2.9 Ensure the default cgroup usage has been confirmed (Scored)

Profile Applicability:

- Level 2 - Docker - Linux

Description:

The `--cgroup-parent` option allows you to set the default cgroup parent to use for all containers. If there is no specific usage requirement for this, the setting should be left at its default.

Rationale:

System administrators typically define cgroups under which containers are supposed to run. Even if cgroups are not explicitly defined by the system administrators, containers run under `docker` cgroup by default.

It is possible to attach to a different cgroup other than the one which is the default, however this type of usage should be monitored and confirmed because attaching to a different cgroup other than the one that is a default, it could be possible to share resources unevenly causing resource utilization problems on the host.

Audit:

In order to confirm this setting, the dockerd start-up options and any settings in `/etc/docker/daemon.json` should be reviewed.
To review the dockerd startup options, use:

```
ps -ef | grep dockerd
```

You should ensure that the `--cgroup-parent` parameter is either not set or is set as appropriate non-default cgroup.
The contents of `/etc/docker/daemon.json` should also be checked for this setting.

Remediation:

The default setting is in line with good security practice and can be left in situ. If you wish to specifically set a non-default cgroup, pass the `--cgroup-parent` parameter to the Docker daemon when starting it.
For example,

```
dockerd --cgroup-parent=/foobar
```

Impact:

None.

Default Value:

By default, docker daemon uses `/docker` for fs cgroup driver and `system.slice` for systemd cgroup driver.

References:

1. https://docs.docker.com/engine/reference/commandline/dockerd/#default-cgroup-parent

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

2.10 Ensure base device size is not changed until needed (Scored)

Profile Applicability:

- Level 2 - Docker - Linux

Description:

Under certain circumstances, you might need containers larger than 10G. Where this applies you should carefully choose the base device size.

Rationale:

The base device size can be increased on daemon restart. Increasing the base device size allows all future images and containers to be of the new base device size. A user can use this option to expand the base device size, however shrinking is not permitted. This value affects the system wide "base" empty filesystem that may already be initialized and therefore inherited by pulled images.

Although the file system does not allocate the increased size as long as it is empty, more space will be allocated for extra images. This may cause a denial of service condition if the allocated partition becomes full.

Audit:

To confirm this setting the dockerd start-up options and any settings in `/etc/docker/daemon.json` should be reviewed.
To review the dockerd startup options, use:

```
ps -ef | grep dockerd
```

Execute the above command and it should not show any `--storage-opt dm.basesize` parameters.
The contents of `/etc/docker/daemon.json` should also be reviewed

Remediation:

Do not set `--storage-opt dm.basesize` until needed.

Impact:

None.

Default Value:

The default base device size is 10G.

References:

1. https://docs.docker.com/engine/reference/commandline/dockerd/#storage-driver-options

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

2.11 Ensure that authorization for Docker client commands is enabled (Scored)

Profile Applicability:

- Level 2 - Docker - Linux

Description:

You should use native Docker authorization plugins or a third party authorization mechanism with the Docker daemon to manage access to Docker client commands.

Rationale:

Docker's out-of-the-box authorization model is currently "all or nothing". This means that any user with permission to access the Docker daemon can run any Docker client command. The same is true for remote users accessing Docker's API to contact the daemon. If you require greater access control, you can create authorization plugins and add them to your Docker daemon configuration. Using an authorization plugin, a Docker administrator can configure granular access policies for managing access to the Docker daemon.

Third party integrations of Docker may implement their own authorization models to require authorization with the Docker daemon outside of docker's native authorization plugin (i.e. Kubernetes, Cloud Foundry, Openshift).

Audit:

To confirm this setting the dockerd start-up options and any settings in `/etc/docker/daemon.json` should be reviewed.
To review the dockerd startup options, use:

```
ps -ef | grep dockerd
```

You should ensure that the `--authorization-plugin` parameter is set as appropriate if you are using docker native authorization.
The contents of `/etc/docker/daemon.json` should also be reviewed.

Remediation:

Step 1: Install/Create an authorization plugin.
Step 2: Configure the authorization policy as desired.
Step 3: Start the docker daemon as below:

```
dockerd --authorization-plugin=<PLUGIN_ID>
```

Impact:

Each Docker command needs to pass through the authorization plugin mechanism. This may have a performance impact.

It may be possible to use alternative mechanisms that do not have this performance hit.

Default Value:

By default, authorization plugins are not set up.

References:

1. https://docs.docker.com/engine/reference/commandline/dockerd/#access-authorization
2. https://docs.docker.com/engine/extend/plugins_authorization/
3. https://github.com/twistlock/authz

Notes:

It should be noted that the native Docker authentication plugin is only one method of enforcing this control so other methods which could potentially be in use should be reviewed before assessing this as a pass or fail in an audit.

CIS Controls:

Version 6

16 Account Monitoring and Control
Account Monitoring and Control

2.12 Ensure centralized and remote logging is configured (Scored)

Profile Applicability:

- Level 2 - Docker - Linux

Description:

Docker supports various logging mechanisms. A preferable method for storing logs is one that supports centralized and remote management.

Rationale:

Centralized and remote logging ensures that all important log records are safe even in the event of a major data availability issue . Docker supports various logging methods and you should use the one that best corresponds to your IT security policy.

Audit:

Run `docker info` and ensure that the `Logging Driver`property set as appropriate.

```
docker info --format '{{ .LoggingDriver }}'
```

Alternatively, the below command would give you the `--log-driver` setting. If configured you should ensure that it is set appropriately.

```
ps -ef | grep dockerd
```

The contents of `/etc/docker/daemon.json` should also be reviewed for this setting.

Remediation:

Step 1: Set up the desired log driver following its documentation.
Step 2: Start the docker daemon using that logging driver.
For example:

```
dockerd --log-driver=syslog --log-opt syslog-address=tcp://192.xxx.xxx.xxx
```

Impact:

None.

Default Value:

By default, container logs are maintained as json files

References:

1. https://docs.docker.com/engine/admin/logging/overview/

CIS Controls:

Version 6

6.6 Deploy A SIEM OR Log Analysis Tools For Aggregation And Correlation/Analysis

Deploy a SIEM (Security Information and Event Management) or log analytic tools for log aggregation and consolidation from multiple machines and for log correlation and analysis. Using the SIEM tool, system administrators and security personnel should devise profiles of common events from given systems so that they can tune detection to focus on unusual activity, avoid false positives, more rapidly identify anomalies, and prevent overwhelming analysts with insignificant alerts.

Version 7

6.6 Deploy SIEM or Log Analytic tool

Deploy Security Information and Event Management (SIEM) or log analytic tool for log correlation and analysis.

6.8 Regularly Tune SIEM

On a regular basis, tune your SIEM system to better identify actionable events and decrease event noise.

2.13 Ensure live restore is enabled (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

The `--live-restore` option enables full support of daemon-less containers within Docker. It ensures that Docker does not stop containers on shutdown or restore and that it properly reconnects to the container when restarted.

Rationale:

One of the important security triads is availability. Setting the `--live-restore` flag within the Docker daemon ensures that container execution is not interrupted when it is not available. This also makes it easier to update and patch the Docker daemon without application downtime.

Audit:

You should run `docker info` and ensure that the `Live Restore Enabled` property is set to `true`.

```
docker info --format '{{ .LiveRestoreEnabled }}'
```

Alternatively, you could run the below command and ensure that `--live-restore` is in use.

```
ps -ef | grep dockerd
```

The contents of `/etc/docker/daemon.json` should also be reviewed to ensure this setting is in place.

Remediation:

Run Docker in daemon mode and pass `--live-restore` to it as an argument.
For Example,

```
dockerd --live-restore
```

Impact:

None.

Default Value:

By default, `--live-restore` is not enabled.

References:

1. https://docs.docker.com/engine/admin/live-restore/

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

2.14 Ensure Userland Proxy is Disabled (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

The Docker daemon starts a userland proxy service for port forwarding whenever a port is exposed. Where hairpin NAT is available, this service is generally superfluous to requirements and can be disabled.

Rationale:

The Docker engine provides two mechanisms for forwarding ports from the host to containers, hairpin NAT, and the use of a userland proxy. In most circumstances, the hairpin NAT mode is preferred as it improves performance and makes use of native Linux iptables functionality instead of using an additional component.

Where hairpin NAT is available, the userland proxy should be disabled on startup to reduce the attack surface of the installation.

Audit:

To confirm this setting, you should review the dockerd start-up options and any settings in `/etc/docker/daemon.json`.
To review the dockerd startup options, use:

```
ps -ef | grep dockerd
```

Ensure that the `--userland-proxy` parameter is set to `false`.
The contents of `/etc/docker/daemon.json` should also be reviewed for this setting.

Remediation:

You should run the Docker daemon as below:

```
dockerd --userland-proxy=false
```

Impact:

Some systems with older Linux kernels may not be able to support hairpin NAT and therefore require the userland proxy service. Also, some networking setups can be impacted by the removal of the userland proxy.

Default Value:

By default, the userland proxy is enabled.

References:

1. http://windsock.io/the-docker-proxy/
2. https://github.com/docker/docker/issues/14856
3. https://github.com/docker/docker/issues/22741
4. https://docs.docker.com/engine/userguide/networking/default_network/binding/

CIS Controls:

Version 6

9.1 Limit Open Ports, Protocols, and Services

Ensure that only ports, protocols, and services with validated business needs are running on each system.

Version 7

9.2 Ensure Only Approved Ports, Protocols and Services Are Running

Ensure that only network ports, protocols, and services listening on a system with validated business needs, are running on each system.

2.15 Ensure that a daemon-wide custom seccomp profile is applied if appropriate (Not Scored)

Profile Applicability:

- Level 2 - Docker - Linux

Description:

You can choose to apply a custom seccomp profile at a daemon-wide level if needed with this overriding Docker's default seccomp profile.

Rationale:

A large number of system calls are exposed to every userland process with many of them not utilized during the entire lifetime of the process. Many applications do not need all these system calls and therefore benefit by having each system call currently in use reviewed in line with organizational security policy. A reduced set of system calls reduces the total kernel surface exposed to the application and therefore improves application security.

A custom seccomp profile can be applied instead of Docker's default seccomp profile. Alternatively, if Docker's default profile is adequate for your environment, you can choose to ignore this recommendation.

Audit:

You should run the command below and review the seccomp profile listed in the `Security Options` section. If it is `default` this indicates that Docker's default seccomp profile is applied.

```
docker info --format '{{ .SecurityOptions }}'
```

Remediation:

By default, Docker's default seccomp profile is applied. If this is adequate for your environment, no action is necessary. Alternatively, if you choose to apply your own seccomp profile, use the `--seccomp-profile` flag at daemon start or put it in the daemon runtime parameters file.

```
dockerd --seccomp-profile </path/to/seccomp/profile>
```

Impact:

A misconfigured seccomp profile could possibly interrupt your container environment. Docker-default blocked calls have been carefully scrutinized and address some critical vulnerabilities/issues within container environments (for example, kernel key ring calls). You should therefore exercise extreme care if you choose to override the default settings.

Default Value:

By default, Docker applies a default seccomp profile.

References:

1. https://docs.docker.com/engine/security/seccomp/
2. https://github.com/docker/docker/pull/26276

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

2.16 Ensure that experimental features are not implemented in production (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

Experimental features should not be enabled in production.

Rationale:

"Experimental" is currently a runtime Docker daemon flag rather than being a feature of a separate build. Passing `--experimental` as a runtime flag to the docker daemon activates experimental features. Whilst "Experimental" is considered a stable release, it has a number of features which may not have been fully tested and do not guarantee API stability.

Audit:

You should run the command below and ensure that the `Experimental` property is set to `false` in the Server section.

```
docker version --format '{{ .Server.Experimental }}'
```

Remediation:

You should not pass `--experimental` as a runtime parameter to the Docker daemon on production systems.

Impact:

None

Default Value:

By default, experimental features are not activated in the Docker daemon.

References:

1. https://docs.docker.com/edge/engine/reference/commandline/dockerd/#options

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

2.17 Ensure containers are restricted from acquiring new privileges (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

By default you should restrict containers from acquiring additional privileges via suid or sgid.

Rationale:

A process can set the `no_new_priv` bit in the kernel and this persists across forks, clones and execve. The `no_new_priv` bit ensures that the process and its child processes do not gain any additional privileges via suid or sgid bits. This reduces the security risks associated with many dangerous operations because there is a much reduced ability to subvert privileged binaries.

Setting this at the daemon level ensures that by default all new containers are restricted from acquiring new privileges.

Audit:

To confirm this setting, you should review the dockerd start-up options and a check of any settings in `/etc/docker/daemon.json` should also be carried out.
To review the dockerd startup options, the following command can be used:

```
ps -ef | grep dockerd
```

You should ensure that the `--no-new-privileges` parameter is present and that it is not set to `false`.
The contents of `/etc/docker/daemon.json` should also be reviewed.

Remediation:

You should run the Docker daemon as below:

```
dockerd --no-new-privileges
```

Impact:

`no_new_priv` prevents LSMs such as SELinux from escalating the privileges of individual containers.

Default Value:

By default, containers are not restricted from acquiring new privileges.

References:

1. https://github.com/moby/moby/pull/29984
2. https://github.com/moby/moby/pull/20727

CIS Controls:

Version 6

5 Controlled Use of Administration Privileges
Controlled Use of Administration Privileges

3 Docker daemon configuration files

This section covers Docker related files and directory permissions and ownership. Keeping the files and directories, that may contain sensitive parameters, secure is important for correct and secure functioning of Docker daemon.

3.1 Ensure that the docker.service file ownership is set to root:root (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the `docker.service` file ownership and group ownership are correctly set to `root`.

Rationale:

The `docker.service` file contains sensitive parameters that may alter the behavior of the Docker daemon. It should therefore be individually and group owned by the `root` user in order to ensure that it is not modified or corrupted by a less privileged user.

Audit:

Step 1: Find out the file location:

```
systemctl show -p FragmentPath docker.service
```

Step 2: If the file does not exist, this recommendation is not applicable. If the file exists, execute the command below including the correct file path in order to verify that the file is owned and group owned by `root`.
For example:

```
stat -c %U:%G /usr/lib/systemd/system/docker.service | grep -v root:root
```

The command above should not return anything.

Remediation:

Step 1: Find out the file location:

```
systemctl show -p FragmentPath docker.service
```

Step 2: If the file does not exist, this recommendation is not applicable. If the file does exist, you should execute the command below, including the correct file path, in order to set the ownership and group ownership for the file to `root`.
For example,

```
chown root:root /usr/lib/systemd/system/docker.service
```

Impact:

None.

Default Value:

This file may not be present on the system and if it is not, this recommendation is not applicable. By default, if the file is present, the correct permissions are for the ownership and group ownership to be set to "root".

References:

1. https://docs.docker.com/engine/admin/systemd/

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

3.2 Ensure that docker.service file permissions are appropriately set (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the `docker.service` file permissions are either set to `644` or to a more restrictive value.

Rationale:

The `docker.service` file contains sensitive parameters that may alter the behavior of the Docker daemon. It should therefore not be writable by any other user other than `root` in order to ensure that it can not be modified by less privileged users.

Audit:

Step 1: Find out the file location:

```
systemctl show -p FragmentPath docker.service
```

Step 2: If the file does not exist, this recommendation is not applicable. If the file exists, execute the command below, including the correct file path in order to verify that the file permissions are set to `644` or a more restrictive value.
For example:

```
stat -c %a /usr/lib/systemd/system/docker.service
```

Remediation:

Step 1: Find out the file location:

```
systemctl show -p FragmentPath docker.service
```

Step 2: If the file does not exist, this recommendation is not applicable. If the file exists, execute the command below including the correct file path to set the file permissions to `644`.
For example,

```
chmod 644 /usr/lib/systemd/system/docker.service
```

Impact:

None.

Default Value:

This file may not be present on the system. In that case, this recommendation is not applicable. By default, if the file is present, the file permissions are correctly set to `644`.

References:

1. https://docs.docker.com/articles/systemd/

CIS Controls:

Version 6

14.4 Protect Information With Access Control Lists
All information stored on systems shall be protected with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

Version 7

14.6 Protect Information through Access Control Lists
Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

3.3 Ensure that docker.socket file ownership is set to root:root (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the `docker.socket` file ownership and group ownership are correctly set to `root`.

Rationale:

The `docker.socket` file contains sensitive parameters that may alter the behavior of the Docker remote API. For this reason, it should be owned and group owned by `root` in order to ensure that it is not modified by less privileged users.

Audit:

Step 1: Find out the file location:

```
systemctl show -p FragmentPath docker.socket
```

Step 2: If the file does not exist, this recommendation is not applicable. If the file exists, execute the command below, including the correct file path to verify that the file is owned and group-owned by `root`.
For example,

```
stat -c %U:%G /usr/lib/systemd/system/docker.socket | grep -v root:root
```

The command above should not return a value.

Remediation:

Step 1: Find out the file location:

```
systemctl show -p FragmentPath docker.socket
```

Step 2: If the file does not exist, this recommendation is not applicable. If the file exists, execute the command below, including the correct file path to set the ownership and group ownership for the file to `root`.
For example,

```
chown root:root /usr/lib/systemd/system/docker.socket
```

Impact:

None.

Default Value:

This file may not be present on the system. In that case, this recommendation is not applicable. By default, if the file is present, the ownership and group ownership for it should be set to `root`.

References:

1. https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-socket-option
2. https://github.com/docker/docker-ce/blob/master/components/packaging/deb/systemd/docker.socket

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

3.4 Ensure that docker.socket file permissions are set to 644 or more restrictive (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the file permissions on the `docker.socket` file are correctly set to `644` or more restrictively.

Rationale:

The `docker.socket` file contains sensitive parameters that may alter the behavior of the Docker remote API. It should therefore be writeable only by `root` in order to ensure that it is not modified by less privileged users.

Audit:

Step 1: Find out the file location:

```
systemctl show -p FragmentPath docker.socket
```

Step 2: If the file does not exist, this recommendation is not applicable. If the file exists, you should execute the command below, including the correct file path in order to verify that the file permissions are set to `644` or more restrictively.
For example:

```
stat -c %a /usr/lib/systemd/system/docker.socket
```

Remediation:

Step 1: Find out the file location:

```
systemctl show -p FragmentPath docker.socket
```

Step 2: If the file does not exist, this recommendation is not applicable. If the file does exist, you should execute the command below, including the correct file path to set the file permissions to `644`.
For example,

```
chmod 644 /usr/lib/systemd/system/docker.socket
```

Impact:

None.

Default Value:

This file may not be present on the system and in that case, this recommendation is not applicable. By default, if the file is present, the permissions should be set to 644 or more restrictively.

References:

1. https://docs.docker.com/engine/reference/commandline/dockerd/#bind-docker-to-another-hostport-or-a-unix-socket
2. https://github.com/YungSang/fedora-atomic-packer/blob/master/oem/docker.socket
3. http://daviddaeschler.com/2014/12/14/centos-7rhel-7-and-docker-containers-on-boot/

CIS Controls:

Version 6

14.4 Protect Information With Access Control Lists

All information stored on systems shall be protected with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

Version 7

14.6 Protect Information through Access Control Lists

Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

3.5 Ensure that the /etc/docker directory ownership is set to root:root (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the `/etc/docker` directory ownership and group ownership is correctly set to `root`.

Rationale:

The `/etc/docker` directory contains certificates and keys in addition to various other sensitive files. It should therefore be individual owned and group owned by `root` in order to ensure that it can not be modified by less privileged users.

Audit:

You should execute the command below to verify that the directory is owned and group owned by `root`:

```
stat -c %U:%G /etc/docker | grep -v root:root
```

This command should not return any data.

Remediation:

To resolve this issue you should run the following command:

```
chown root:root /etc/docker
```

This sets the ownership and group ownership for the directory to `root`.

Impact:

None.

Default Value:

By default, the ownership and group ownership for this directory is correctly set to `root`.

References:

1. https://docs.docker.com/engine/security/https/

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

3.6 Ensure that /etc/docker directory permissions are set to 755 or more restrictively (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the `/etc/docker` directory permissions are correctly set to `755` or more restrictively.

Rationale:

The `/etc/docker` directory contains certificates and keys in addition to various sensitive files. It should therefore only be writeable by `root` to ensure that it can not be modified by a less privileged user.

Audit:

You should execute the command below to verify that the directory has permissions of `755` or more restrictive ones:

```
stat -c %a /etc/docker
```

Remediation:

You should run the following command:

```
chmod 755 /etc/docker
```

This sets the permissions for the directory to `755`.

Impact:

None.

Default Value:

By default, the permissions for this directory are set to `755`.

References:

1. https://docs.docker.com/engine/security/https/

CIS Controls:

Version 6

14.4 Protect Information With Access Control Lists

All information stored on systems shall be protected with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

Version 7

14.6 Protect Information through Access Control Lists

Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

3.7 Ensure that registry certificate file ownership is set to root:root (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that all the registry certificate files (usually found under `/etc/docker/certs.d/<registry-name>` directory) are individually owned and group owned by `root`.

Rationale:

The `/etc/docker/certs.d/<registry-name>` directory contains Docker registry certificates. These certificate files must be individually owned and group owned by `root` to ensure that less privileged users are unable to modify the contents of the directory.

Audit:

You should execute the command below to verify that the registry certificate files are individually owned and group owned by `root`:

```
stat -c %U:%G /etc/docker/certs.d/* | grep -v root:root
```

The above command should not return any value.

Remediation:

The following command could be executed:

```
chown root:root /etc/docker/certs.d/<registry-name>/*
```

This would set the individual ownership and group ownership for the registry certificate files to `root`.

Impact:

None.

Default Value:

By default, the individual ownership and group ownership for registry certificate files is correctly set to `root`.

References:

1. https://docs.docker.com/registry/insecure/

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

3.8 Ensure that registry certificate file permissions are set to 444 or more restrictively (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that all the registry certificate files (usually found under `/etc/docker/certs.d/<registry-name>` directory) have permissions of `444` or are set more restrictively.

Rationale:

The `/etc/docker/certs.d/<registry-name>` directory contains Docker registry certificates. These certificate files must have permissions of `444`or more restrictive permissions in order to ensure that unprivileged users do not have full access to them..

Audit:

You should execute the command below to verify that registry certificate files have permissions of `444` or are more restrictively set.

```
stat -c %a /etc/docker/certs.d/<registry-name>/*
```

Remediation:

You should execute the following command:

```
chmod 444 /etc/docker/certs.d/<registry-name>/*
```

This would set the permissions for the registry certificate files to `444`.

Impact:

None.

Default Value:

By default, the permissions for registry certificate files might not be `444`. The default file permissions are governed by the system or user specific `umask`values which are defined within the operating system itself.

References:

1. https://docs.docker.com/registry/insecure/

CIS Controls:

Version 6

14.4 Protect Information With Access Control Lists

All information stored on systems shall be protected with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

Version 7

14.6 Protect Information through Access Control Lists

Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

3.9 Ensure that TLS CA certificate file ownership is set to root:root (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the TLS CA certificate file (the file that is passed along with the `--tlscacert` parameter) is individually owned and group owned by `root`.

Rationale:

The TLS CA certificate file should be protected from any tampering. It is used to authenticate the Docker server based on a given CA certificate. It must be therefore be individually owned and group owned by `root` to ensure that it cannot be modified by less privileged users.

Audit:

You should execute the command below to verify that the TLS CA certificate file is owned and group owned by `root`:

```
stat -c %U:%G <path to TLS CA certificate file> | grep -v root:root
```

The above command should return no results.

Remediation:

You should execute the following command:

```
chown root:root <path to TLS CA certificate file>
```

This sets the individual ownership and group ownership for the TLS CA certificate file to `root`.

Impact:

None.

Default Value:

By default, the ownership and group-ownership for TLS CA certificate file is correctly set to `root`.

References:

1. https://docs.docker.com/registry/insecure/
2. https://docs.docker.com/engine/security/https/

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

3.10 Ensure that TLS CA certificate file permissions are set to 444 or more restrictively (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the TLS CA certificate file (the file that is passed along with the `--tlscacert` parameter) has permissions of `444` or is set more restrictively.

Rationale:

The TLS CA certificate file should be protected from any tampering. It is used to authenticate the Docker server based on a given CA certificate. It must therefore have permissions of `444`, or more restrictive permissions to ensure that the file cannot be modified by a less privileged user.

Audit:

You should execute the command below to verify that the TLS CA certificate file has permissions of `444` or that these are more restrictively set:

```
stat -c %a <path to TLS CA certificate file>
```

Remediation:

You should execute the following command:

```
chmod 444 <path to TLS CA certificate file>
```

This sets the file permissions on the TLS CA file to `444`.

Impact:

None.

Default Value:

By default, the permissions for the TLS CA certificate file might not be `444`. The default file permissions are governed by the operating system or user specific `umask` values.

References:

1. https://docs.docker.com/registry/insecure/
2. https://docs.docker.com/engine/security/https/

CIS Controls:

Version 6

14.4 Protect Information With Access Control Lists

All information stored on systems shall be protected with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

Version 7

14.6 Protect Information through Access Control Lists

Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

3.11 Ensure that Docker server certificate file ownership is set to root:root (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the Docker server certificate file (the file that is passed along with the `--tlscert` parameter) is individual owned and group owned by `root`.

Rationale:

The Docker server certificate file should be protected from any tampering. It is used to authenticate the Docker server based on the given server certificate. It must therefore be individually owned and group owned by `root` to prevent modification by less privileged users.

Audit:

You should execute the command below to verify that the Docker server certificate file is individually owned and group owned by `root`:

```
stat -c %U:%G <path to Docker server certificate file> | grep -v root:root
```

The above command should return no results.

Remediation:

You should run the following command:

```
chown root:root <path to Docker server certificate file>
```

This sets the individual ownership and the group ownership for the Docker server certificate file to `root`.

Impact:

None.

Default Value:

By default, the ownership and group-ownership for Docker server certificate file is correctly set to `root`.

References:

1. https://docs.docker.com/registry/insecure/
2. https://docs.docker.com/engine/security/https/

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

3.12 Ensure that the Docker server certificate file permissions are set to 444 or more restrictively (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the Docker server certificate file (the file that is passed along with the `--tlscert` parameter) has permissions of `444` or more restrictive permissions.

Rationale:

The Docker server certificate file should be protected from any tampering. It is used to authenticate the Docker server based on the given server certificate. It should therefore have permissions of `444` to prevent its modification.

Audit:

You should execute the command below to verify that the Docker server certificate file has permissions of `444` or more restrictive permissions:

```
stat -c %a <path to Docker server certificate file>
```

Remediation:

You should execute the command below:

```
chmod 444 <path to Docker server certificate file>
```

This sets the file permissions of the Docker server certificate file to `444`.

Impact:

None.

Default Value:

By default, the permissions for the Docker server certificate file might not be `444`. The default file permissions are governed by the operating system or user specific `umask` values.

References:

1. https://docs.docker.com/registry/insecure/

2. https://docs.docker.com/engine/security/https/

CIS Controls:

Version 6

14.4 Protect Information With Access Control Lists

All information stored on systems shall be protected with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

Version 7

14.6 Protect Information through Access Control Lists

Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

3.13 Ensure that the Docker server certificate key file ownership is set to root:root (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the Docker server certificate key file (the file that is passed along with the `--tlskey` parameter) is individually owned and group owned by `root`.

Rationale:

The Docker server certificate key file should be protected from any tampering or unneeded reads/writes. As it holds the private key for the Docker server certificate, it must be individually owned and group owned by `root` to ensure that it cannot be accessed by less privileged users.

Audit:

You should execute the command below to verify that the Docker server certificate key file is individually owned and group owned by `root`:

```
stat -c %U:%G <path to Docker server certificate key file> | grep -v
root:root
```

The command above should return no results.

Remediation:

You should execute the following command:

```
chown root:root <path to Docker server certificate key file>
```

This sets the individual ownership and group ownership for the Docker server certificate key file to `root`.

Impact:

None.

Default Value:

By default, the individual ownership and group ownership for the Docker server certificate key file is correctly set to `root`.

References:

1. https://docs.docker.com/registry/insecure/
2. https://docs.docker.com/engine/security/https/

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

3.14 Ensure that the Docker server certificate key file permissions are set to 400 (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the Docker server certificate key file (the file that is passed along with the `--tlskey` parameter) has permissions of `400`.

Rationale:

The Docker server certificate key file should be protected from any tampering or unneeded reads. It holds the private key for the Docker server certificate. It must therefore have permissions of `400` to ensure that the certificate key file is not modified.

Audit:

You should execute the command below to verify that the Docker server certificate key file has permissions of `400`:

```
stat -c %a <path to Docker server certificate key file>
```

Remediation:

You should execute the following command:

```
chmod 400 <path to Docker server certificate key file>
```

This sets the Docker server certificate key file permissions to `400`.

Impact:

None.

Default Value:

By default, the permissions for the Docker server certificate key file might not be `400`. The default file permissions are governed by the operating system or user specific `umask` values.

References:

1. https://docs.docker.com/registry/insecure/

2. https://docs.docker.com/engine/security/https/

CIS Controls:

Version 6

14.4 Protect Information With Access Control Lists

All information stored on systems shall be protected with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

Version 7

14.6 Protect Information through Access Control Lists

Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

3.15 Ensure that the Docker socket file ownership is set to root:docker (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the Docker socket file is owned by `root` and group owned by `docker`.

Rationale:

The Docker daemon runs as `root`. The default Unix socket therefore must be owned by `root`. If any other user or process owns this socket, it might be possible for that non-privileged user or process to interact with the Docker daemon. Additionally, in this case a non-privileged user or process might be able to interact with containers which is neither a secure nor desired behavior.

Additionally, the Docker installer creates a Unix group called `docker`. You can add users to this group, and in this case, those users would be able to read and write to the default Docker Unix socket. The membership of the `docker` group is tightly controlled by the system administrator. However, ff any other group owns this socket, then it might be possible for members of that group to interact with the Docker daemon. Such a group might not be as tightly controlled as the `docker` group. Again, this is not in line with good security practice.

For these reason, the default Docker Unix socket file should be owned by `root` and group owned by `docker` to maintain the integrity of the socket file.

Audit:

You should execute the below command to verify that the Docker socket file is owned by `root` and group owned by `docker`:

```
stat -c %U:%G /var/run/docker.sock | grep -v root:docker
```

The command above should return no results.

Remediation:

You should execute the following command:

```
chown root:docker /var/run/docker.sock
```

This sets the ownership to `root` and group ownership to `docker` for the default Docker socket file.

Impact:

None.

Default Value:

By default, the ownership and group ownership for the Docker socket file is correctly set to `root:docker`.

References:

1. https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-socket-option
2. https://docs.docker.com/engine/reference/commandline/dockerd/#bind-docker-to-another-hostport-or-a-unix-socket

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

3.16 Ensure that the Docker socket file permissions are set to 660 or more restrictively (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the Docker socket file has permissions of `660` or are configured more restrictively.

Rationale:

Only `root` and the members of the `docker` group should be allowed to read and write to the default Docker Unix socket. The Docker socket file should therefore have permissions of `660` or more restrictive permissions.

Audit:

You should execute the command below to verify that the Docker socket file has permissions of `660` or more restrictive permissions

```
stat -c %a /var/run/docker.sock
```

Remediation:

You should execute the command below.

```
chmod 660 /var/run/docker.sock
```

This sets the file permissions of the Docker socket file to `660`.

Impact:

None.

Default Value:

By default, the permissions for the Docker socket file is correctly set to `660`.

References:

1. https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-socket-option

2. https://docs.docker.com/engine/reference/commandline/dockerd/#bind-docker-to-another-hostport-or-a-unix-socket

CIS Controls:

Version 6

14.4 Protect Information With Access Control Lists

All information stored on systems shall be protected with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

Version 7

14.6 Protect Information through Access Control Lists

Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

3.17 Ensure that the daemon.json file ownership is set to root:root (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the `daemon.json` file individual ownership and group ownership is correctly set to `root`.

Rationale:

The `daemon.json` file contains sensitive parameters that could alter the behavior of the docker daemon. It should therefore be owned and group owned by `root` to ensure it can not be modified by less privileged users.

Audit:

You should execute the command below to verify that the file is owned and group owned by `root`:

```
stat -c %U:%G /etc/docker/daemon.json | grep -v root:root
```

The command above should not return any results.

Remediation:

You should execute the command below:

```
chown root:root /etc/docker/daemon.json
```

This sets the ownership and group ownership for the file to `root`.

Impact:

None.

Default Value:

This file may not be present on the system, and in that case, this recommendation is not applicable.

References:

1. https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

3.18 Ensure that daemon.json file permissions are set to 644 or more restrictive (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the `daemon.json` file permissions are correctly set to `644` or more restrictively.

Rationale:

The `daemon.json` file contains sensitive parameters that may alter the behavior of the docker daemon. Therefore it should be writeable only by `root` to ensure it is not modified by less privileged users.

Audit:

You should execute the command below to verify that the file permissions are correctly set to `644` or more restrictively:

```
stat -c %a /etc/docker/daemon.json
```

Remediation:

You should execute the command below

```
chmod 644 /etc/docker/daemon.json
```

This sets the file permissions for this file to `644`.

Impact:

None.

Default Value:

This file may not be present on the system, and in that case, this recommendation is not applicable.

References:

1. https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file

CIS Controls:

Version 6

14.4 Protect Information With Access Control Lists

All information stored on systems shall be protected with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

Version 7

14.6 Protect Information through Access Control Lists

Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

3.19 Ensure that the /etc/default/docker file ownership is set to root:root (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the `/etc/default/docker` file ownership and group-ownership is correctly set to `root`.

Rationale:

The `/etc/default/docker` file contains sensitive parameters that may alter the behavior of the Docker daemon. It should therefore be individually owned and group owned by `root` to ensure that it cannot be modified by less privileged users.

Audit:

You should execute the command below to verify that the file is individually owned and group owned by `root`:

```
stat -c %U:%G /etc/default/docker | grep -v root:root
```

The command above should return no results.

Remediation:

You should execute the following command

```
chown root:root /etc/default/docker
```

This sets the ownership and group ownership of the file to `root`.

Impact:

None.

Default Value:

This file may not be present on the system, and in this case, this recommendation is not applicable.

References:

1. https://docs.docker.com/engine/admin/configuring/

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

3.20 Ensure that the /etc/sysconfig/docker file ownership is set to root:root (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the `/etc/sysconfig/docker` file individual ownership and group ownership is correctly set to `root`.

Rationale:

The `/etc/sysconfig/docker` file contains sensitive parameters that may alter the behavior of the Docker daemon. It should therefore be individually owned and group owned by `root` to ensure that it is not modified by less privileged users.

Audit:

You should execute the command below to verify that the file is indiviually owned and group owned by `root`:

```
stat -c %U:%G /etc/sysconfig/docker | grep -v root:root
```

The command above should return no results.

Remediation:

You should execute the following command:

```
chown root:root /etc/sysconfig/docker
```

This sets the ownership and group ownership for the file to `root`.

Impact:

None.

Default Value:

This file may not be present on the system, and in this case, this recommendation is not applicable.

References:

1. https://docs.docker.com/engine/admin/configuring/

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

3.21 Ensure that the /etc/sysconfig/docker file permissions are set to 644 or more restrictively (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the `/etc/sysconfig/docker` file permissions are correctly set to `644` or more restrictively.

Rationale:

The `/etc/sysconfig/docker` file contains sensitive parameters that may alter the behavior of the Docker daemon. It should therefore be writeable only by `root` in order to ensure that it is not modified by less privileged users.

Audit:

You should execute the command below to verify that the file permissions are correctly set to `644` or more restrictively:

```
stat -c %a /etc/sysconfig/docker
```

Remediation:

You should execute the following command:

```
chmod 644 /etc/sysconfig/docker
```

This sets the file permissions for this file to `644`.

Impact:

None.

Default Value:

This file may not be present on the system and in this case, this recommendation is not applicable.

References:

1. https://docs.docker.com/engine/admin/configuring/

CIS Controls:

Version 6

14.4 Protect Information With Access Control Lists

All information stored on systems shall be protected with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

Version 7

14.6 Protect Information through Access Control Lists

Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

3.22 Ensure that the /etc/default/docker file permissions are set to 644 or more restrictively (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should verify that the `/etc/default/docker` file permissions are correctly set to `644` or more restrictively.

Rationale:

The `/etc/default/docker` file contains sensitive parameters that may alter the behavior of the Docker daemon. It should therefore be writeable only by `root` in order to ensure that it is not modified by less privileged users.

Audit:

You should execute the command below to verify that the file permissions are correctly set to `644` or more restrictively:

```
stat -c %a /etc/default/docker
```

Remediation:

You should execute the following command:

```
chmod 644 /etc/default/docker
```

This sets the file permissions for this file to `644`.

Impact:

None.

Default Value:

This file may not be present on the system and in this case, this recommendation is not applicable.

References:

1. https://docs.docker.com/engine/admin/configuring/

CIS Controls:

Version 6

14.4 Protect Information With Access Control Lists

All information stored on systems shall be protected with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

Version 7

14.6 Protect Information through Access Control Lists

Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

4 Container Images and Build File Configuration

Container base images and build files govern the fundamentals of how a container instance from a particular image would behave. Ensuring that you are using proper base images and appropriate build files can be very important for building your containerized infrastructure. Below are some of the recommendations that you should follow for container base images and build files to ensure that your containerized infrastructure is secure.

4.1 Ensure that a user for the container has been created (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

Containers should run as a non-root user.

Rationale:

It is good practice to run the container as a non-root user, where possible. This can be done either via the `USER` directive in the `Dockerfile` or through `gosu` or similar where used as part of the `CMD` or `ENTRYPOINT` directives.

Audit:

You should run the following command

```
docker ps --quiet | xargs --max-args=1 -I{} docker exec {} cat /proc/1/status
| grep '^Uid:' | awk '{print $3}'
```

This should return the effective UID for each container and where it returns 0, it indicates that the container process is running as root.

Remediation:

You should ensure that the Dockerfile for each container image contains the information below:

```
USER <username or ID>
```

In this case, the user name or ID refers to the user that was found in the container base image. If there is no specific user created in the container base image, then make use of the

useradd command to add a specific user before the USER instruction in the Dockerfile. For example, add the below lines in the Dockerfile to create a user in the container:

```
RUN useradd -d /home/username -m -s /bin/bash username
USER username
```

Note: If there are users in the image that are not needed, you should consider deleting them. After deleting those users, commit the image and then generate new instances of the containers.
Alternatively, if it is not possible to set the USER directive in the Dockerfile, a script running as part of the CMD or ENTRYPOINT sections of the Dockerfile should be used to ensure that the container process switches to a non-root user.

Impact:

Running as a non-root user can present challenges where you wish to bind mount volumes from the underlying host. In this case, care should be taken to ensure that the user running the contained process can read and write to the bound directory, according to their requirements.

Default Value:

By default, containers are run with rootprivileges and also run as the root user inside the container.

References:

1. https://github.com/docker/docker/issues/2918
2. https://github.com/docker/docker/pull/4572
3. https://github.com/docker/docker/issues/7906

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

4.2 Ensure that containers use only trusted base images (Not Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should ensure that container images you use are either written from scratch or are based on another established and trusted base image downloaded over a secure channel.

Rationale:

Official repositories contain Docker images curated and optimized by the Docker community or by their vendor. There is no guarantee that these images are safe and do not contain security vulnerabilities or malicious code. Caution should therefore be exercised when obtaining container images from Docker and third parties and running these images should be reviewed in line with organizational security policy.

Audit:

You should review what Docker images are present on the host by executing the command below:

```
docker images
```

This command lists all the container images that are currently available for use on the Docker host. You should then review the origin of each image and review its contents in line with your organization's security policy.
You can use the command below to review the history of commits to the image.

```
docker history <imageName>
```

Remediation:

The following procedures are useful for establishing trust for a specific image.

- Configure and use Docker Content trust.
- View the history of each Docker image to evaluate its risk, dependent on the sensitivity of the application you wish to deploy using it.
- Scan Docker images for vulnerabilities at regular intervals.

Impact:

None.

Default Value:

Not Applicable.

References:

1. https://titanous.com/posts/docker-insecurity
2. https://registry.hub.docker.com/
3. http://blog.docker.com/2014/10/docker-1-3-signed-images-process-injection-security-options-mac-shared-directories/
4. https://github.com/docker/docker/issues/8093
5. https://docs.docker.com/engine/reference/commandline/pull/
6. https://github.com/docker/docker/pull/11109
7. https://blog.docker.com/2015/11/docker-trusted-registry-1-4/

CIS Controls:

Version 6

3 Secure Configurations for Hardware and Software on Mobile Devices, Laptops, Workstations, and Servers

Secure Configurations for Hardware and Software on Mobile Devices, Laptops, Workstations, and Servers

4.3 Ensure that unnecessary packages are not installed in the container (Not Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

Containers should have as small a footprint as possible, and should not contain unnecessary software packages which could increase their attack surface.

Rationale:

Unnecessary software should not be installed into containers, as doing so increases their attack surface. Only packages strictly necessary for the correct operation of the application being deployed should be installed.

Audit:

Step 1: List all the running instances of containers by executing the command below:

```
docker ps --quiet
```

Step 2: For each container instance, execute the command below:

```
docker exec $INSTANCE_ID rpm -qa
```

The command above lists the packages installed. You should review the list and ensure that everything installed is actually required.

Remediation:

You should not install anything within the container that is not required.
You should consider using a minimal base image rather than the standard Redhat/Centos/Debian images if you can. Some of the options available include BusyBox and Alpine.
Not only can this trim your image size considerably, but there would also be fewer pieces of software which could contain vectors for attack.

Impact:

None.

Default Value:

Not Applicable.

References:

1. https://docs.docker.com/userguide/dockerimages/
2. http://www.livewyer.com/blog/2015/02/24/slimming-down-your-docker-containers-alpine-linux
3. https://github.com/progrium/busybox

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

4.4 Ensure images are scanned and rebuilt to include security patches (Not Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

Images should be scanned frequently for any vulnerabilities. You should rebuild all images to include these patches and then instantiate new containers from them.

Rationale:

Vulnerabilities are loopholes or bugs that can be exploited by hackers or malicious users, and security patches are updates to resolve these vulnerabilities. Image vulnerability scanning tools can be use to find vulnerabilities in images and then check for available patches to mitigate these. Patches update the system to a more recent code base which does not contain these problems, and being on a supported version of the code base is very important, as vendors do not tend to supply patches for older versions which have gone out of support. Security patches should be evaluated before applying and patching should be implemented in line with the organization's IT Security Policy.

Care should be taken with the results returned by vulnerability assessment tools, as some will simply return results based on software banners, and these may not be entirely accurate.

Audit:

Step 1: List all the running instances of containers by executing the command below:

```
docker ps --quiet
```

Step 2: For each container instance, use the package manager within the container (assuming there is one available) to check for the availability of security patches. Alternatively, run image vulnerability assessment tools to scan all the images in your environment.

Remediation:

Images should be re-built ensuring that the latest version of the base images are used, to keep the operating system patch level at an appropriate level. Once the images have been re-built, containers should be re-started making use of the updated images.

Impact:

None

Default Value:

By default, containers and images are not updated automatically to address missing operating system security patches..

References:

1. https://docs.docker.com/userguide/dockerimages/
2. https://docs.docker.com/docker-cloud/builds/image-scan/
3. https://blog.docker.com/2016/05/docker-security-scanning/
4. https://docs.docker.com/engine/reference/builder/#/onbuild

CIS Controls:

Version 6

18.1 Use Only Vendor-supported Software

For all acquired application software, check that the version you are using is still supported by the vendor. If not, update to the most current version and install all relevant patches and vendor security recommendations.

Version 7

18.3 Verify That Acquired Software is Still Supported

Verify that the version of all software acquired from outside your organization is still supported by the developer or appropriately hardened based on developer security recommendations.

4.5 Ensure Content trust for Docker is Enabled (Scored)

Profile Applicability:

- Level 2 - Docker - Linux

Description:

Content trust is disabled by default and should be enabled in line with organizational security policy.

Rationale:

Content trust provides the ability to use digital signatures for data sent to and received from remote Docker registries. These signatures allow client-side verification of the identity and the publisher of specific image tags and ensures the provenance of container images.

Audit:

You should execute the following command:

```
echo $DOCKER_CONTENT_TRUST
```

This should return a value of 1.

Remediation:

To enable content trust in a bash shell, you should enter the following command:

```
export DOCKER_CONTENT_TRUST=1
```

Alternatively, you could set this environment variable in your profile file so that content trust in enabled on every login.

Impact:

In an environment where `DOCKER_CONTENT_TRUST` is set, you are required to follow trust procedures whilst working with the image related commands - `build`, `create`, `pull`, `push`and `run`. You can use the `--disable-content-trust` flag to run individual operations on tagged images without content trust on an as needed basis, but this defeats the purpose of enabling content trust and therefore should be avoided wherever possible.

Note: Content trust is currently only available for users of the public Docker Hub. It is currently not available for the Docker Trusted Registry or for private registries.

Default Value:

By default, content trust is disabled.

References:

1. https://docs.docker.com/engine/security/trust/content_trust/
2. https://docs.docker.com/engine/reference/commandline/cli/#notary
3. https://docs.docker.com/engine/reference/commandline/cli/#environment-variables

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

4.6 Ensure that HEALTHCHECK instructions have been added to container images (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should add the `HEALTHCHECK` instruction to your Docker container images in order to ensure that health checks are executed against running containers.

Rationale:

An important security control is that of availability. Adding the `HEALTHCHECK` instruction to your container image ensures that the Docker engine periodically checks the running container instances against that instruction to ensure that containers are still operational.

Based on the results of the health check, the Docker engine could terminate containers which are not responding correctly, and instantiate new ones.

Audit:

You should run the command below to ensure that Docker images have the appropriate `HEALTHCHECK` instruction configured.

```
docker inspect --format='{{ .Config.Healthcheck }}' <IMAGE>
```

Remediation:

You should follow the Docker documentation and rebuild your container images to include the `HEALTHCHECK` instruction.

Impact:

None.

Default Value:

By default, `HEALTHCHECK` is not set.

References:

1. https://docs.docker.com/engine/reference/builder/#healthcheck

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

4.7 Ensure update instructions are not use alone in the Dockerfile (Not Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should not use OS package manager update instructions such as `apt-get update` or `yum update` either alone or in a single line in the Dockerfile.

Rationale:

Adding update instructions in a single line on the Dockerfile will cause the update layer to be cached. When you then build any image later using the same instruction, this will cause the previously cached update layer to be used, potentially preventing any fresh updates from being applied to later builds.

Audit:

Step 1: Run the command below to get the list of images:

```
docker images
```

Step 2: Run the command below against each image in the list above, looking for any update instructions which are incorporated in a single line:

```
docker history <Image_ID>
```

Alternatively, if you have access to the Dockerfile for the image, you should verify that there are no update instructions configured as described above.

Remediation:

You should use update instructions together with install instructions and version pinning for packages while installing them. This prevent caching and force the extraction of the required versions.
Alternatively, you could use the `--no-cache` flag during the `docker build` process to avoid using cached layers.

Impact:

None

Default Value:

By default, Docker does not enforce any restrictions on using update instructions.

References:

1. https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#run
2. https://github.com/docker/docker/issues/3313

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

4.8 Ensure setuid and setgid permissions are removed (Not Scored)

Profile Applicability:

- Level 2 - Docker - Linux

Description:

Removing setuid and setgid permissions in the images can prevent privilege escalation attacks within containers.

Rationale:

setuid and setgid permissions can be used for privilege escalation. Whilst these permissions can on occasion be legitimately needed, you should consider removing them from packages which do not need them. This should be reviewed for each image.

Audit:

You should run the command below against each image to list the executables which have either setuid or setgid permissions:

```
docker run <Image_ID> find / -perm /6000 -type f -exec ls -ld {} \; 2>
/dev/null
```

You should then review the list and ensure that all executables configured with these permissions actually require them.

Remediation:

You should allow setuid and setgid permissions only on executables which require them. You could remove these permissions at build time by adding the following command in your Dockerfile, preferably towards the end of the Dockerfile:

```
RUN find / -perm /6000 -type f -exec chmod a-s {} \; || true
```

Impact:

The above command would break all executables that depend on setuid or setgid permissions including legitimate ones. You should therefore be careful to modify the command to suit your requirements so that it does not reduce the permissions of legitimate programs excessively. Because of this, you should exercise a degree of caution and examine all processes carefully before making this type of modification in order to avoid outages.

Default Value:

Not Applicable

References:

1. http://www.oreilly.com/webops-perf/free/files/docker-security.pdf
2. http://container-solutions.com/content/uploads/2015/06/15.06.15_DockerCheatSheet_A2.pdf
3. http://man7.org/linux/man-pages/man2/setuid.2.html
4. http://man7.org/linux/man-pages/man2/setgid.2.html

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

4.9 Ensure that COPY is used instead of ADD in Dockerfiles (Not Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should use the `COPY` instruction instead of the `ADD` instruction in the Dockerfile.

Rationale:

The `COPY` instruction simply copies files from the local host machine to the container file system. The `ADD` instruction could potentially retrieve files from remote URLs and perform operations such as unpacking them. The `ADD` instruction therefore introduces security risks. For example, malicious files may be directly accessed from URLs without scanning, or there may be vulnerabilities associated with decompressing them..

Audit:

Step 1: Run the command below to get the list of images:

```
docker images
```

Step 2: Run the command below against each image in the list above and look for any `ADD` instructions:

```
docker history <Image_ID>
```

Alternatively, if you have access to the Dockerfile for the image, you should verify that there are no `ADD` instructions.

Remediation:

You should use `COPY` rather than `ADD` instructions in Dockerfiles.

Impact:

Care needs to be taken in implementing this control if the application requires functionality that is part of the `ADD` instruction, for example, if you need to retrieve files from remote URLS.

Default Value:

Not Applicable

References:

1. https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#add-or-copy

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

4.10 Ensure secrets are not stored in Dockerfiles (Not Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

Do not store any secrets in Dockerfiles.

Rationale:

Docker images are not opaque and contain information about the commands used to build them. As such secrets should not be included in Dockerfiles used to build images as they will be visible to any users of the image.

Audit:

Step 1: Run the below command to get the list of images:

```
docker images
```

Step 2: Run the below command for each image in the list above, and look for any secrets:

```
docker history <Image_ID>
```

Alternatively, if you have access to Dockerfile for the image, verify that there are no secrets as described above.

Remediation:

Do not store any kind of secrets within Dockerfiles. Where secrets are required during the build process, make use of a secrets management tool, such as the buildkit builder included with Docker.

Impact:

A proper secrets management process will be required for Docker image building.

Default Value:

By default, there are no restrictions on storing config secrets in the Dockerfiles.

References:

1. https://github.com/docker/docker/issues/13490

2. http://12factor.net/config
3. https://avicoder.me/2016/07/22/Twitter-Vine-Source-code-dump/
4. https://docs.docker.com/develop/develop-images/build_enhancements/

CIS Controls:

Version 6

14 Controlled Access Based on the Need to Know
Controlled Access Based on the Need to Know

4.11 Ensure only verified packages are are installed (Not Scored)

Profile Applicability:

- Level 2 - Docker - Linux

Description:

You should verify the authenticity of packages before installing them into images.

Rationale:

Verifying authenticity of software packages is essential for building a secure container image. Packages with no known provenance could potentially be malicious or have vulnerabilities that could be exploited.

Audit:

Step 1: Run the command below to get the list of images:

```
docker images
```

Step 2: Run the command below for each image in the list above, and check how the authenticity of the packages is being determined. This could be via the use of GPG keys or other secure package distribution mechanisms.

```
docker history <Image_ID>
```

Alternatively, if you have access to Dockerfile for the image, verify that the authenticity of the packages is checked.

Remediation:

You should use a secure package distribution mechanism of your choice to ensure the authenticity of software packages.

Impact:

None

Default Value:

Not Applicable

References:

1. http://www.oreilly.com/webops-perf/free/files/docker-security.pdf
2. https://github.com/docker-library/httpd/blob/12bf8c8883340c98b3988a7bade8ef2d0d6dcf8a/2.4/Dockerfile
3. https://github.com/docker-library/php/blob/d8a4ccf4d620ec866d5b42335b699742df08c5f0/7.0/alpine/Dockerfile
4. https://access.redhat.com/security/team/key

CIS Controls:

Version 6

18.1 Use Only Vendor-supported Software

For all acquired application software, check that the version you are using is still supported by the vendor. If not, update to the most current version and install all relevant patches and vendor security recommendations.

Version 7

18.3 Verify That Acquired Software is Still Supported

Verify that the version of all software acquired from outside your organization is still supported by the developer or appropriately hardened based on developer security recommendations.

5 Container Runtime Configuration

There are many security implications associated with the ways that containers are started. Some runtime parameters can be supplied that have security consequences that could compromise the host and the containers running on it. It is therefore very important to verify the way in which containers are started, and which parameters are associated with them. Container runtime configuration should be reviewed in line with organizational security policy.

5.1 Ensure that, if applicable, an AppArmor Profile is enabled (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

AppArmor is an effective and easy-to-use Linux application security system. It is available on some Linux distributions by default, for example, on Debian and Ubuntu.

Rationale:

AppArmor protects the Linux OS and applications from various threats by enforcing a security policy which is also known as an AppArmor profile. You can create your own AppArmor profile for containers or use Docker's default profile. Enabling this feature enforces security policies on containers as defined in the profile.

Audit:

You should run the command below:

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
AppArmorProfile={{ .AppArmorProfile }}'
```

This command should return a valid AppArmor Profile for each container instance.

Remediation:

If AppArmor is applicable for your Linux OS, you should enable it.

1. Verify AppArmor is installed.
2. Create or import a AppArmor profile for Docker containers.
3. Enable enforcement of the policy.
4. Start your Docker container using the customized AppArmor profile. For example:

```
docker run --interactive --tty --security-opt="apparmor:PROFILENAME" ubuntu
/bin/bash
```

Alternatively, Docker's default AppArmor policy can be used.

Impact:

The container will have the security controls defined in the AppArmor profile. It should be noted that if the AppArmor profile is misconfigured, this may cause issues with the operation of the container.

Default Value:

By default, the `docker-default` AppArmor profile is applied to running containers. This profile can be found at `/etc/apparmor.d/docker`.

References:

1. https://docs.docker.com/engine/security/apparmor/
2. https://docs.docker.com/engine/reference/run/#security-configuration
3. https://docs.docker.com/engine/security/security/#other-kernel-security-features

CIS Controls:

Version 6

14.4 Protect Information With Access Control Lists

All information stored on systems shall be protected with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

Version 7

14.6 Protect Information through Access Control Lists

Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

5.2 Ensure that, if applicable, SELinux security options are set (Scored)

Profile Applicability:

- Level 2 - Docker - Linux

Description:

SELinux is an effective and easy-to-use Linux application security system. It is available by default on some distributions such as Red Hat and Fedora.

Rationale:

SELinux provides a Mandatory Access Control (MAC) system that greatly augments the default Discretionary Access Control (DAC) model. You can therefore add an extra layer of safety to your containers by enabling SELinux on your Linux host.

Audit:

You should run the following command

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
SecurityOpt={{ .HostConfig.SecurityOpt }}'
```

This command returns all the security options currently configured on the containers listed.

Remediation:

If SELinux is applicable for your Linux OS, you should use it.

1. Set the SELinux State.
2. Set the SELinux Policy.
3. Create or import a SELinux policy template for Docker containers.
4. Start Docker in daemon mode with SELinux enabled. For example:

```
docker daemon --selinux-enabled
```

5. Start your Docker container using the security options. For example,

```
docker run --interactive --tty --security-opt label=level:TopSecret centos
/bin/bash
```

Impact:

Any restrictions defined in the SELinux policy will be applied to your containers. It should be noted that if your SELinux policy is misconfigured, this may have an impact on the correct operation of the affected containers.

Default Value:

By default, no SELinux security options are applied on containers.

References:

1. https://docs.docker.com/engine/security/security/#other-kernel-security-features
2. https://docs.docker.com/engine/reference/run/#security-configuration
3. http://docs.fedoraproject.org/en-US/Fedora/13/html/Security-Enhanced_Linux/
4. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/container_security_guide/docker_selinux_security_policy

CIS Controls:

Version 6

14.4 Protect Information With Access Control Lists

All information stored on systems shall be protected with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

Version 7

14.6 Protect Information through Access Control Lists

Protect all information stored on systems with file system, network share, claims, application, or database specific access control lists. These controls will enforce the principle that only authorized individuals should have access to the information based on their need to access the information as a part of their responsibilities.

5.3 Ensure that Linux kernel capabilities are restricted within containers (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

By default, Docker starts containers with a restricted set of Linux kernel capabilities. This means that any process can be granted the required capabilities instead of giving it root access. Using Linux kernel capabilities, processes in general do not need to run as the root user.

Rationale:

Docker supports the addition and removal of capabilities. You should remove all capabilities not required for the correct function of the container.

Specifically, in the default capability set provided by Docker, the `NET_RAW` capability should be removed if not explicitly required, as it can give an attacker with access to a container the ability to create spoofed network traffic.

Audit:

You should run the following command:

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: CapAdd={{
.HostConfig.CapAdd }} CapDrop={{ .HostConfig.CapDrop }}'
```

Verify that the added and deleted Linux kernel capabilities are in line with the ones needed by the container process in each container instance. Specifically, ensure that the `NET_RAW` capability is removed if not required.

Remediation:

You should execute the command below to add required capabilities:

```
docker run --cap-add={"Capability 1","Capability 2"} <Run arguments>
<Container Image Name or ID> <Command>
```

You should execute the command below to remove unneeded capabilities:

```
docker run --cap-drop={"Capability 1","Capability 2"} <Run arguments>
<Container Image Name or ID> <Command>
```

Alternatively, you could remove all the currently configured capabilities and then restore only the ones you specifically use:

```
docker run --cap-drop=all --cap-add={"Capability 1","Capability 2"} <Run
arguments> <Container Image Name or ID> <Command>
```

Impact:

Restrictions on processes within a container are based on which Linux capabilities are in force. Removal of the `NET_RAW` capability prevents the container from creating raw sockets which is good security practice under most circumstances, but may affect some networking utilities.

Default Value:

By default, the capabilities below are applied to containers:

```
AUDIT_WRITE

CHOWN

DAC_OVERRIDE

FOWNER

FSETID

KILL

MKNOD

NET_BIND_SERVICE

NET_RAW

SETFCAP

SETGID

SETPCAP

SETUID

SYS_CHROOT
```

References:

1. https://docs.docker.com/engine/security/security/#linux-kernel-capabilities
2. http://man7.org/linux/man-pages/man7/capabilities.7.html
3. http://www.oreilly.com/webops-perf/free/files/docker-security.pdf

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

5.4 Ensure that privileged containers are not used (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

Using the `--privileged` flag provides all Linux kernel capabilities to the container to which it is applied and therefore overwrites the `--cap-add` and `--cap-drop` flags. For this reason you should ensure that it is not used.

Rationale:

The `--privileged` flag provides all capabilities to the container to which it is applied, and also lifts all the limitations enforced by the device cgroup controller. As a consequence this the container has most of the rights of the underlying host. This flag only exists to allow for specific use cases (for example running Docker within Docker) and should not generally be used.

Audit:

You should run the command below:

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
Privileged={{ .HostConfig.Privileged }}'
```

The above command should return `Privileged=false` for each container instance.

Remediation:

You should not run containers with the `--privileged` flag.
For example, do not start a container using the command below:

```
docker run --interactive --tty --privileged centos /bin/bash
```

Impact:

If you start a container without the `--privileged` flag, it will not have excessive default capabilities.

Default Value:

False.

References:

1. https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

5.5 Ensure sensitive host system directories are not mounted on containers (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should not allow sensitive host system directories such as those listed below to be mounted as container volumes, especially in read-write mode.

```
/

/boot

/dev

/etc

/lib

/proc

/sys

/usr
```

Rationale:

If sensitive directories are mounted in read-write mode, it could be possible to make changes to files within them. This has obvious security implications and should be avoided.

Audit:

You should run the following command:

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
Volumes={{ .Mounts }}'
```

This command returns a list of currently mapped directories and indicates whether they are mounted in read-write mode for each container instance.

Remediation:

You should not mount directories which are security sensitive on the host within containers, especially in read-write mode.

Impact:

None.

Default Value:

Docker defaults to using a read-write volume but you can also mount a directory read-only. By default, no sensitive host directories are mounted within containers.

References:

1. https://docs.docker.com/engine/tutorials/dockervolumes/

CIS Controls:

Version 6

14 Controlled Access Based on the Need to Know
Controlled Access Based on the Need to Know

5.6 Ensure sshd is not run within containers (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

The SSH daemon should not be running within the container. You should SSH into the Docker host, and use `docker exec` to enter a container.

Rationale:

Running SSH within the container increases the complexity of security management by making it

- Difficult to manage access policies and security compliance for SSH server
- Difficult to manage keys and passwords across various containers
- Difficult to manage security upgrades for SSH server

It is possible to have shell access to a container without using SSH, the needlessly increasing the complexity of security management should be avoided.

Audit:

Step 1: List all the running instances of containers by executing below command:

```
docker ps --quiet
```

Step 2: For each container instance, execute the below command:

```
docker exec $INSTANCE_ID ps -el
```

Ensure that there is no process for SSH server.

Remediation:

Uninstall the SSH daemon from the container and use and use `docker exec` to enter a container on the remote host.

```
docker exec --interactive --tty $INSTANCE_ID sh
```

OR

```
docker attach $INSTANCE_ID
```

Impact:

None.

Default Value:

By default, SSH server is not running inside the container. Only one process per container is allowed.

References:

1. http://blog.docker.com/2014/06/why-you-dont-need-to-run-sshd-in-docker/

CIS Controls:

Version 6

9.1 Limit Open Ports, Protocols, and Services
Ensure that only ports, protocols, and services with validated business needs are running on each system.

Version 7

9.2 Ensure Only Approved Ports, Protocols and Services Are Running
Ensure that only network ports, protocols, and services listening on a system with validated business needs, are running on each system.

5.7 Ensure privileged ports are not mapped within containers (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

The TCP/IP port numbers below `1024`are considered privileged ports. Normal users and processes are not allowed to use them for various security reasons. Docker does, however allow a container port to be mapped to a privileged port.

Rationale:

By default, if the user does not specifically declare a container port to host port mapping, Docker automatically and correctly maps the container port to one available in the `49153-65535` range on the host. Docker does, however, allow a container port to be mapped to a privileged port on the host if the user explicitly declares it. This is because containers are executed with `NET_BIND_SERVICE` Linux kernel capability which does not restrict privileged port mapping. The privileged ports receive and transmit various pieces of data which are security sensitive and allowing containers to use them is not in line with good security practice.

Audit:

You can list all running containers instances and their port mappings by executing the command below:

```
docker ps --quiet | xargs docker inspect --format '{{ .Id }}: Ports={{
.NetworkSettings.Ports }}'
```

You should then review the list and ensure that container ports are not mapped to host port numbers below `1024`.

Remediation:

You should not map container ports to privileged host ports when starting a container. You should also, ensure that there is no such container to host privileged port mapping declarations in the Dockerfile.

Impact:

None.

Default Value:

By default, mapping a container port to a privileged port on the host is allowed.

Note: There might be certain cases where you want to map privileged ports, because if you forbid it, then the corresponding application has to run outside of a container.

For example: HTTP and HTTPS load balancers have to bind 80/tcp and 443/tcp respectively. Forbidding to map privileged ports effectively forbids from running those in a container, and mandates using an external load balancer. In such cases, those containers instances should be marked as exceptions for this recommendation.

References:

1. https://docs.docker.com/engine/userguide/networking/

CIS Controls:

Version 6

9.1 Limit Open Ports, Protocols, and Services
Ensure that only ports, protocols, and services with validated business needs are running on each system.

Version 7

9.2 Ensure Only Approved Ports, Protocols and Services Are Running
Ensure that only network ports, protocols, and services listening on a system with validated business needs, are running on each system.

5.8 Ensure that only needed ports are open on the container (Not Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

The dockerfile for a container image defines the ports which are opened by default on a container instance. The list of ports are relevant to the application you are running within the container and should only be open if they are needed.

Rationale:

A container can be run with only the ports defined in the Dockerfile for its image or can alternatively be arbitrarily passed run time parameters to open a list of ports. Additionally, in the course of time, the Dockerfile may undergo various changes and the list of exposed ports may or may not still be relevant to the application you are running within the container. Opening unneeded ports increases the attack surface of the container and the associated containerized application. Good security practice is to only open ports that are needed for the correct operation of the application.

Audit:

You should list all the running instances of containers and their associated port mappings by executing the command below:

```
docker ps --quiet | xargs docker inspect --format '{{ .Id }}: Ports={{
.NetworkSettings.Ports }}'
```

You should then review the list and ensure that all the ports mapped are in fact genuinely required by each container.

Remediation:

You should ensure that the Dockerfile for each container image only exposes needed ports. You can also completely ignore the list of ports defined in the Dockerfile by **NOT** using `-P` (UPPERCASE) or the `--publish-all` flag when starting the container. Instead, use the `-p` (lowercase) or `--publish` flag to explicitly define the ports that you need for a particular container instance.
For example:

```
docker run --interactive --tty --publish 5000 --publish 5001 --publish 5002
centos /bin/bash
```

Impact:

None.

Default Value:

By default, all the ports that are listed in the Dockerfile under the `EXPOSE` instruction for an image are opened when a container is run with the `-P` or `--publish-all` flags.

References:

1. https://docs.docker.com/engine/userguide/networking/

CIS Controls:

Version 6

9.1 Limit Open Ports, Protocols, and Services
Ensure that only ports, protocols, and services with validated business needs are running on each system.

Version 7

9.2 Ensure Only Approved Ports, Protocols and Services Are Running
Ensure that only network ports, protocols, and services listening on a system with validated business needs, are running on each system.

5.9 Ensure that the host's network namespace is not shared (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

When the networking mode on a container is set to `--net=host`, the container is not placed inside a separate network stack. Effectively, applying this option instructs Docker to not containerize the container's networking. The consequence of this is that the container lives "outside" in the main Docker host and has full access to its network interfaces.

Rationale:

Selecting this option is potentially dangerous. It allows the container process to open reserved low numbered ports in the way that any other `root` process can. It also allows the container to access network services such as D-bus on the Docker host. A container process could potentially carry out undesired actions, such as shutting down the Docker host. This option should not be used unless there is a very specific reason for enabling it.

Audit:

You should use the command below:

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
NetworkMode={{ .HostConfig.NetworkMode }}'
```

If this returns `NetworkMode=host`, it means that the `--net=host` option was passed when the container was started.

Remediation:

You should not pass the `--net=host` option when starting any container.

Impact:

None.

Default Value:

By default, containers connect to the Docker `bridge` when starting and do not run in the context of the host's network stack.

References:

1. https://docs.docker.com/engine/userguide/networking/
2. https://docs.docker.com/engine/reference/run/#network-settings

CIS Controls:

Version 6

12 Boundary Defense
Boundary Defense

5.10 Ensure that the memory usage for containers is limited (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

By default, all containers on a Docker host share resources equally. By using the resource management capabilities of the Docker host, you can control the amount of memory that a container is able to use.

Rationale:

By default a container can use all of the memory on the host. You can use memory limit mechanisms to prevent a denial of service occurring where one container consumes all of the host's resources and other containers on the same host are therefore not able to function. Having no limit on memory usage can lead to issues where one container can easily make the whole system unstable and as a result unusable.

Audit:

You should run the command below:

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: Memory={{
.HostConfig.Memory }}'
```

If this command returns `0`, it means that memory limits are not in place; if it returns a non-zero value, it means that they are in place.

Remediation:

You should run the container with only as much memory as it requires by using the `--memory` argument.
For example, you could run a container using the command below:

```
docker run --interactive --tty --memory 256m centos /bin/bash
```

In the example above, the container is started with a memory limit of 256 MB.
Note that the output of the command below returns values in scientific notation if memory limits are in place.

```
docker inspect --format='{{.Config.Memory}}' 7c5a2d4c7fe0
```

For example, if the memory limit is set to `256 MB` for a container instance, the output of the command above would be `2.68435456e+08` and NOT 256m. You should convert this value using a scientific calculator.

Impact:

If correct memory limits are not set on each container, one process can expand its usage and cause other containers to run out of resources.

Default Value:

By default, all containers on a Docker host share their resources equally and no memory limits are enforced.

References:

1. https://goldmann.pl/blog/2014/09/11/resource-management-in-docker/
2. https://docs.docker.com/engine/reference/commandline/run/#options
3. https://docs.docker.com/engine/admin/runmetrics/

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

5.11 Ensure that CPU priority is set appropriately on containers (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

By default, all containers on a Docker host share resources equally. By using the resource management capabilities of the Docker host you can control the host CPU resources that a container may consume.

Rationale:

By default, CPU time is divided between containers equally. If you wish to control available CPU resources amongst container instances, you can use the CPU sharing feature. CPU sharing allows you to prioritize one container over others and prevents lower priority containers from absorbing CPU resources which may be required by other processes. This ensures that high priority containers are able to claim the CPU runtime they require.

Audit:

You should run the following command.

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
CpuShares={{ .HostConfig.CpuShares }}'
```

If the above command returns `0` or `1024`, it means that CPU shares are not in place. If it returns a non-zero value other than `1024`, it means that they are in place.

Remediation:

You should manage the CPU runtime between your containers dependent on their priority within your organization. To do so start the container using the `--cpu-shares` argument. For example, you could run a container as below:

```
docker run --interactive --tty --cpu-shares 512 centos /bin/bash
```

In the example above, the container is started with CPU shares of 50% of what other containers use. So if the other container has CPU shares of 80%, this container will have CPU shares of 40%.
Every new container will have `1024` shares of CPU by default. However, this value is shown as `0` if you run the command mentioned in the audit section.
Alternatively:

1. Navigate to the `/sys/fs/cgroup/cpu/system.slice/` directory.
2. Check your container instance ID using `docker ps`.
3. Inside the above directory (in step 1), you could have a directory called, for example: `docker-<Instance ID>.scope`. For example, `docker-4acae729e8659c6be696ee35b2237cc1fe4edd2672e9186434c5116e1a6fbed6.scope`. Navigate to this directory.
4. You will find a file named `cpu.shares`. Execute `cat cpu.shares`. This will always give you the CPU share value based on the system. Even if there are no CPU shares configured using the `-c` or `--cpu-shares` argument in the `docker run` command, this file will have a value of `1024`.

If you set one container's CPU shares to `512` it will receive half of the CPU time compared to the other containers. So if you take `1024` as 100% you can then derive the number that you should set for respective CPU shares. For example, use `512` if you want to set it to 50% and `256` if you want to set it 25%.

Impact:

If you do not correctly assign CPU thresholds, the container process may run out of resources and become unresponsive. If CPU resources on the host are not constrainted, CPU shares do not place any restrictions on individual resources.

Default Value:

By default, all containers on a Docker host share their resources equally. No CPU shares are enforced.

References:

1. https://goldmann.pl/blog/2014/09/11/resource-management-in-docker/
2. https://docs.docker.com/engine/reference/commandline/run/#options
3. https://docs.docker.com/engine/admin/runmetrics/

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

5.12 Ensure that the container's root filesystem is mounted as read only (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

The container's root filesystem should be treated as a 'golden image' by using Docker run's `--read-only` option. This prevents any writes to the container's root filesystem at container runtime and enforces the principle of immutable infrastructure.

Rationale:

Enabling this option forces containers at runtime to explicitly define their data writing strategy to persist or not persist their data.

This also reduces security attack vectors since the container instance's filesystem cannot be tampered with or written to unless it has explicit read-write permissions on its filesystem folder and directories.

Audit:

You should run the following command on the docker host:

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
ReadonlyRootfs={{ .HostConfig.ReadonlyRootfs }}'
```

If the above command returns `true`, it means the container's root filesystem is mounted read-only.
If the above command returns `false`, it means the container's root filesystem is writeable.

Remediation:

You should add a `--read-only` flag at a container's runtime to enforce the container's root filesystem being mounted as read only.

```
docker run <Run arguments> --read-only <Container Image Name or ID> <Command>
```

Enabling the `--read-only` option at a container's runtime should be used by administrators to force a container's executable processes to only write container data to explicit storage locations during its lifetime.

Examples of explicit storage locations during a container's runtime include, but are not limited to:

1. Using the `--tmpfs` option to mount a temporary file system for non-persistent data writes.

```
docker run --interactive --tty --read-only --tmpfs "/run" --tmpfs "/tmp"
centos /bin/bash
```

2. Enabling Docker `rw` mounts at a container's runtime to persist container data directly on the Docker host filesystem.

```
docker run --interactive --tty --read-only -v /opt/app/data:/run/app/data:rw
centos /bin/bash
```

3. Utilizing the Docker shared-storage volume plugin for Docker data volume to persist container data.

```
docker volume create -d convoy --opt o=size=20GB my-named-volume

docker run --interactive --tty --read-only -v my-named-volume:/run/app/data
centos /bin/bash
```

3. Transmitting container data outside of the Docker controlled area during the container's runtime for container data in order to ensure that it is persistent. Examples include hosted databases, network file shares and APIs.

Impact:

Enabling `--read-only` at container runtime may break some container OS packages if a data writing strategy is not defined.

You should define what the container's data should and should not persist at runtime in order to decide which strategy to use.

Example:

- Enable use `--tmpfs` for temporary file writes to `/tmp`
- Use Docker shared data volumes for persistent data writes

Default Value:

By default, a container has its root filesystem writeable, allowing all container processes to write files owned by the container's actual runtime user.

References:

1. http://docs.docker.com/reference/commandline/cli/#run
2. https://docs.docker.com/engine/tutorials/dockervolumes/
3. http://www.projectatomic.io/blog/2015/12/making-docker-images-write-only-in-production/
4. https://docs.docker.com/engine/reference/commandline/run/#mount-tmpfs-tmpfs
5. https://docs.docker.com/engine/tutorials/dockervolumes/#creating-and-mounting-a-data-volume-container

CIS Controls:

Version 6

14 Controlled Access Based on the Need to Know
Controlled Access Based on the Need to Know

5.13 Ensure that incoming container traffic is bound to a specific host interface (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

By default, Docker containers can make connections to the outside world, but the outside world cannot connect to containers and each outgoing connection will appear to originate from one of the host machine's own IP addresses. You should only allow container services to be contacted through a specific external interface on the host machine.

Rationale:

If you have multiple network interfaces on your host machine, the container can accept connections on exposed ports on any network interface. This might not be desirable and may not be secured. In many cases a specific, desired interface is exposed externally and services such as intrusion detection, intrusion prevention, firewall, load balancing, etc. are all run by intention there to screen incoming public traffic. You should therefore not accept incoming connections on any random interface, but only the one designated for this type of traffic.

Audit:

You should list all running instances of containers and their port mappings by executing the command below:

```
docker ps --quiet | xargs docker inspect --format '{{ .Id }}: Ports={{
.NetworkSettings.Ports }}'
```

Then review the list and ensure that the exposed container ports are bound to a specific interface and not to the wildcard IP address - `0.0.0.0`.
For example, if the command above returns the results below, this is non-compliant and the container can accept connections on any host interface on the specified port `49153`.

```
Ports=map[443/tcp:<nil> 80/tcp:[map[HostPort:49153 HostIp:0.0.0.0]]]
```

However, if the exposed port is bound to a specific interface on the host as below, then this is configured in line with good security practice.

```
Ports=map[443/tcp:<nil> 80/tcp:[map[HostIp:10.2.3.4 HostPort:49153]]]
```

Remediation:

You should bind the container port to a specific host interface on the desired host port. For example,

```
docker run --detach --publish 10.2.3.4:49153:80 nginx
```

In the example above, the container port `80` is bound to the host port on `49153` and would accept incoming connection only from the `10.2.3.4` external interface.

Impact:

None.

Default Value:

By default, Docker exposes the container ports on `0.0.0.0`, the wildcard IP address that will match any possible incoming network interface on the host machine.

References:

1. https://docs.docker.com/engine/userguide/networking/

CIS Controls:

Version 6

9 Limitation and Control of Network Ports, Protocols, and Services
Limitation and Control of Network Ports, Protocols, and Services

5.14 Ensure that the 'on-failure' container restart policy is set to '5' (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

By using the `--restart` flag in the `docker run` command you can specify a restart policy for how a container should or should not be restarted on exit. You should choose the `on-failure` restart policy and limit the restart attempts to `5`.

Rationale:

If you indefinitely keep trying to start the container, it could possibly lead to a denial of service on the host. It could be an easy way to do a distributed denial of service attack especially if you have many containers on the same host. Additionally, ignoring the exit status of the container and always attempting to restart the container, leads to non-investigation of the root cause behind containers getting terminated. If a container gets terminated, you should investigate on the reason behind it instead of just attempting to restart it indefinitely. You should use the `on-failure` restart policy to limit the number of container restarts to a maximum of `5` attempts.

Audit:

You should use the command below

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
RestartPolicyName={{ .HostConfig.RestartPolicy.Name }} MaximumRetryCount={{
.HostConfig.RestartPolicy.MaximumRetryCount }}'
```

If this command returns `RestartPolicyName=always`, then the system is not configured optimally.
If the above command returns `RestartPolicyName=no` or just `RestartPolicyName=`, then restart policies are not being used and the container would never be restarted automatically. Whilst this may be a secure option, it is not the best option from a usability standpoint.
If the above command returns `RestartPolicyName=on-failure`, then verify that the number of restart attempts is set to `5` or less by looking at `MaximumRetryCount`.

Remediation:

If you wish a container to be automatically restarted, a sample command is as below:

```
docker run --detach --restart=on-failure:5 nginx
```

Impact:

If this option is set, a container will only attempt to restart itself 5 times.

Default Value:

By default, containers are not configured with restart policies.

References:

1. https://docs.docker.com/engine/reference/commandline/run/#restart-policies-restart

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

5.15 Ensure that the host's process namespace is not shared (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

The Process ID (PID) namespace isolates the process ID space, meaning that processes in different PID namespaces can have the same PID. This creates process level isolation between the containers and the host.

Rationale:

PID namespace provides separation between processes. It prevents system processes from being visible, and allows process ids to be reused including PID `1`. If the host's PID namespace is shared with containers, it would basically allow these to see all of the processes on the host system. This reduces the benefit of process level isolation between the host and the containers. Under these circumstances a malicious user who has access to a container could get access to processes on the host itself, manipulate them, and even be able to kill them. This could allow for the host itself being shut down, which could be extremely serious, particularly in a multi-tenanted environment. You should not share the host's process namespace with the containers running on it.

Audit:

You should run the following command:

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
PidMode={{ .HostConfig.PidMode }}'
```

If the command above returns `host`, it means that the host PID namespace is shared with its containers; any other result means that the system is configured in line with good security practice.

Remediation:

You should not start a container with the `--pid=host` argument.
For example, do not start a container with the command below:

```
docker run --interactive --tty --pid=host centos /bin/bash
```

Impact:

Container processes cannot see processes on the host system. In certain circumstances, you may want your container to share the host's process namespace. For example, you could build a container containing debugging tools such as `strace` or `gdb`, and want to use these tools when debugging processes on the host. If this is desired, then share specific host processes using the `-p` switch.

For example:

```
docker run --pid=host rhel7 strace -p 1234
```

Default Value:

By default, all containers have the PID namespace enabled and the therefore the host's process namespace is not shared with its containers.

References:

1. https://docs.docker.com/engine/reference/run/#pid-settings-pid
2. http://man7.org/linux/man-pages/man7/pid_namespaces.7.html

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

5.16 Ensure that the host's IPC namespace is not shared (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

IPC (POSIX/SysV IPC) namespace provides separation of named shared memory segments, semaphores and message queues. The IPC namespace on the host should therefore not be shared with containers and should remain isolated.

Rationale:

The IPC namespace provides separation of IPC between the host and containers. If the host's IPC namespace is shared with the container, it would allow processes within the container to see all of IPC communications on the host system. This would remove the benefit of IPC level isolation between host and containers. An attacker with access to a container could get access to the host at this level with major consequences. The IPC namespace should therefore not be shared between the host and its containers.

Audit:

You should run the following command:

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
IpcMode={{ .HostConfig.IpcMode }}'
```

If the command returns `host`, it means that the host IPC namespace is shared with the container. Any other result means that it is not shared, and that the system is therefore configured in line with good security practice.

Remediation:

You should not start a container with the `--ipc=host` argument. For example, do not start a container as below:

```
docker run --interactive --tty --ipc=host centos /bin/bash
```

Impact:

Shared memory segments are used in order to accelerate interprocess communications, commonly in high-performance applications. If this type of application is containerized into multiple containers, you might need to share the IPC namespace of the containers in order

to achieve high performance. Under these circumstances, you should still only share container specific IPC namespaces and not the host IPC namespace.

A container's IPC namespace can be shared with another container as shown below:

```
docker run --interactive --tty --ipc=container:e3a7a1a97c58 centos /bin/bash
```

Default Value:

By default, all containers have their IPC namespace enabled and host IPC namespace is not shared with any container.

References:

1. https://docs.docker.com/engine/reference/run/#ipc-settings-ipc
2. http://man7.org/linux/man-pages/man7/namespaces.7.html

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

5.17 Ensure that host devices are not directly exposed to containers (Not Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

Host devices can be directly exposed to containers at runtime. Do not directly expose host devices to containers, especially to containers that are not trusted.

Rationale:

The `--device` option exposes host devices to containers and as a result of this, containers can directly access these devices. The the container would not need to run in `privileged` mode to access and manipulate them, as by default, the container is granted this type of access. Additionally, it would possible for containers to remove block devices from the host. You therefore should not expose host devices to containers directly.

If for some reason you wish to expose the host device to a container you should consider which sharing permissions you wish to use on a case by case base as appropriate to your organization:

- r - read only
- w - writable
- m - mknod allowed

Audit:

You should use the command below:

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
Devices={{ .HostConfig.Devices }}'
```

The above command would list out each device with below information:

- `CgroupPermissions` - For example, `rwm`
- `PathInContainer` - Device path within the container
- `PathOnHost` - Device path on the host

You should verify that the host device is needed to be accessed from within the container and that the permissions required are correctly set. If the above command returns [], then the container does not have access to host devices and is configured in line with good security practice.

Remediation:

You should not directly expose host devices to containers. If you do need to expose host devices to containers, you should use granular permissions as appropriate to your organization:
For example, do not start a container using the command below:

```
docker run --interactive --tty --device=/dev/tty0:/dev/tty0:rwm --
device=/dev/temp_sda:/dev/temp_sda:rwm centos bash
```

You should only share the host device using appropriate permissions:

```
docker run --interactive --tty --device=/dev/tty0:/dev/tty0:rw --
device=/dev/temp_sda:/dev/temp_sda:r centos bash
```

Impact:

You would not be able to use host devices directly within containers.

Default Value:

By default, host devices are not exposed to containers. If you do not provide sharing permissions and choose to expose a host device to a container, the host device is be exposed with `read`, `write` and `mknod` permissions.

References:

1. https://docs.docker.com/engine/reference/commandline/run/#options

CIS Controls:

Version 6

14 Controlled Access Based on the Need to Know
Controlled Access Based on the Need to Know

5.18 Ensure that the default ulimit is overwritten at runtime if needed (Not Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

The default ulimit is set at the Docker daemon level. However, if you need to, you may override the default ulimit setting during container runtime.

Rationale:

`ulimit` provides control over the resources available to the shell and to processes started by it. Setting system resource limits in a prudent fashion, protects against denial of service conditions. On occasion, legitimate users and processes can accidentally overuse system resources and cause systems be degraded or even unresponsive.

The default ulimit set at the Docker daemon level should be honored. If the default ulimit settings are not appropriate for a particular container instance, you may override them as an exception, but this should not be done routinely. If many of your container instances are exceeding your ulimit settings, you should consider changing the default settings to something that is more appropriate for your needs.

Audit:

You should run the command below:

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
Ulimits={{ .HostConfig.Ulimits }}'
```

This command should return `Ulimits=<no value>` for each container instance unless there is a need in a specific case to override the default settings.

Remediation:

You should only override the default ulimit settings if needed in a specific case.
For example, to override default ulimit settings start a container as below:

```
docker run --ulimit nofile=1024:1024 --interactive --tty centos /bin/bash
```

Impact:

If ulimits are not set correctly, overutilization by individual containers could make the host system unusable.

Default Value:

Container instances inherit the default ulimit settings set at the Docker daemon level.

References:

1. https://docs.docker.com/engine/reference/commandline/run/#set-ulimits-in-container-ulimit
2. http://www.oreilly.com/webops-perf/free/files/docker-security.pdf

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

5.19 Ensure mount propagation mode is not set to shared (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

Mount propagation mode allows mounting volumes in shared, slave or private mode on a container. Do not use shared mount propagation mode unless explicitly needed.

Rationale:

A shared mount is replicated at all mounts and changes made at any mount point are propagated to all other mount points.

Mounting a volume in shared mode does not restrict any other container from mounting and making changes to that volume.

As this is likely not a desirable option from a security standpoint, this feature should not be used unless explicitly required.

Audit:

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
Propagation={{range $mnt := .Mounts}} {{json $mnt.Propagation}} {{end}}'
```

The above command returns the propagation mode for mounted volumes. The propagation mode should not be set to `shared` unless needed. The above command might throw errors if there are no mounts. In that case, this recommendation is not applicable.

Remediation:

Do not mount volumes in shared mode propagation.
For example, do not start a container as below:

```
docker run <Run arguments> --volume=/hostPath:/containerPath:shared
<Container Image Name or ID> <Command>
```

Impact:

None.

Default Value:

By default, the container mounts are private.

References:

1. https://docs.docker.com/storage/bind-mounts/#configure-bind-propagation
2. https://docs.docker.com/engine/reference/run/#volume-shared-filesystems
3. https://www.kernel.org/doc/Documentation/filesystems/sharedsubtree.txt

CIS Controls:

Version 6

14 Controlled Access Based on the Need to Know
Controlled Access Based on the Need to Know

5.20 Ensure that the host's UTS namespace is not shared (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

UTS namespaces provide isolation between two system identifiers: the hostname and the NIS domain name. It is used to set the hostname and the domain which are visible to running processes in that namespace. Processes running within containers do not typically require to know either the hostname or the domain name. The UTS namespace should therefore not be shared with the host.

Rationale:

Sharing the UTS namespace with the host provides full permission for each container to change the hostname of the host. This is not in line with good security practice and should not be permitted.

Audit:

You should run the following command:

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
UTSMode={{ .HostConfig.UTSMode }}'
```

If the above command returns `host`, it means the host UTS namespace is shared with the container and this recommendation is non-compliant. If the above command returns nothing, then the host's UTS namespace is not shared. This recommendation is then compliant.

Remediation:

You should not start a container with the `--uts=host` argument.
For example, do not start a container using the command below:

```
docker run --rm --interactive --tty --uts=host rhel7.2
```

Impact:

None.

Default Value:

By default, all containers have the UTS namespace enabled and the host UTS namespace is not shared with any containers.

References:

1. https://docs.docker.com/engine/reference/run/#uts-settings-uts
2. http://man7.org/linux/man-pages/man7/namespaces.7.html

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

5.21 Ensure the default seccomp profile is not Disabled (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

Seccomp filtering provides a means for a process to specify a filter for incoming system calls. The default Docker seccomp profile works on a whitelist basis and allows for a large number of common system calls, whilst blocking all others. This filtering should not be disabled unless it causes a problem with your container application usage.

Rationale:

A large number of system calls are exposed to every userland process with many of them going unused for the entire lifetime of the process. Most of applications do not need all these system calls and would therefore benefit from having a reduced set of available system calls. Having a reduced set of system calls reduces the total kernel surface exposed to the application and thus improvises application security.

Audit:

You should run the following command:

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
SecurityOpt={{ .HostConfig.SecurityOpt }}'
```

This should return either `<no value>` or your modified seccomp profile. If it returns `[seccomp:unconfined]`, the container is running without any seccomp profiles and is therefore not configured in line with good security practice.

Remediation:

By default, seccomp profiles are enabled. You do not need to do anything unless you want to modify and use a modified seccomp profile.

Impact:

With Docker 1.10 and greater, the default seccomp profile blocks syscalls, regardless of `--cap-add` passed to the container. You should create your own custom seccomp profile in such cases. You may also disable the default seccomp profile by passing `--security-opt=seccomp:unconfined` on `docker run`.

Default Value:

When you run a container, it uses the default profile unless you override it with the `--security-opt` option.

References:

1. http://blog.scalock.com/new-docker-security-features-and-what-they-mean-seccomp-profiles
2. https://docs.docker.com/engine/reference/run/#security-configuration
3. https://github.com/docker/docker/blob/master/profiles/seccomp/default.json
4. https://docs.docker.com/engine/security/seccomp/
5. https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt
6. https://github.com/docker/docker/issues/22870

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

5.22 Ensure that docker exec commands are not used with the privileged option (Scored)

Profile Applicability:

- Level 2 - Docker - Linux

Description:

You should not use `docker exec` with the `--privileged` option.

Rationale:

Using the `--privileged` option in `docker exec` commands gives extended Linux capabilities to the command. This could potentially be an insecure practice, particularly when you are running containers with reduced capabilities or with enhanced restrictions.

Audit:

If you have auditing enabled as recommended in Section 1, you can use the command below to filter out `docker exec` commands that use the `--privileged` option.

```
ausearch -k docker | grep exec | grep privileged
```

Remediation:

You should not use the `--privileged` option in `docker exec` commands.

Impact:

If you need enhanced capabilities within a container, then run it with all the permissions it requires. These should be specified individually.

Default Value:

By default, the `docker exec` command runs without the `--privileged` option.

References:

1. https://docs.docker.com/engine/reference/commandline/exec/

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges

Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

5.23 Ensure that docker exec commands are not used with the user=root option (Not Scored)

Profile Applicability:

- Level 2 - Docker - Linux

Description:

You should not use `docker exec` with the `--user=root` option.

Rationale:

Using the `--user=root` option in a `docker exec` command, executes it within the container as the root user. This could potentially be insecure, particularly when you are running containers with reduced capabilities or enhanced restrictions.

For example, if your container is running as a tomcat user (or any other non-root user), it would be possible to run a command through `docker exec` as `root`with the `--user=root` option. This could potentially be dangerous.

Audit:

If you have auditing enabled as recommended in Section 1, you can use the command below to filter out `docker exec` commands that use the `--user=root` option.

```
ausearch -k docker | grep exec | grep user
```

Remediation:

You should not use the `--user=root` option in `docker exec` commands.

Impact:

None.

Default Value:

By default, the `docker exec` command runs without the `--user` option.

References:

1. https://docs.docker.com/engine/reference/commandline/exec/

CIS Controls:

Version 6

5 Controlled Use of Administration Privileges
Controlled Use of Administration Privileges

5.24 Ensure that cgroup usage is confirmed (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

It is possible to attach to a particular cgroup when a container is instantiated. Confirming cgroup usage would ensure that containers are running in defined cgroups.

Rationale:

System administrators typically define cgroups in which containers are supposed to run. If cgroups are not explicitly defined by the system administrator, containers run in the `docker` cgroup by default.

At run time, it is possible to attach a container to a different cgroup other than the one originally defined. This usage should be monitored and confirmed, as by attaching to a different cgroup, excess permissions and resources might be granted to the container and this can therefore prove to be a security risk.

Audit:

You should run the following command:

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
CgroupParent={{ .HostConfig.CgroupParent }}'
```

The above command returns the cgroup where the containers are running. If it is blank, it means that containers are running under the default docker cgroup. Any other return value indicates that the system is not configured in line with good security practice.

Remediation:

You should not use the `--cgroup-parent` option within the `docker run` command unless strictly required.

Impact:

None.

Default Value:

By default, containers run under `docker` cgroup.

References:

1. https://docs.docker.com/engine/reference/run/#specify-custom-cgroups
2. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/ch01.html

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

5.25 Ensure that the container is restricted from acquiring additional privileges (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should restrict the container from acquiring additional privileges via suid or sgid bits.

Rationale:

A process can set the `no_new_priv` bit in the kernel and this persists across forks, clones and execve. The `no_new_priv` bit ensures that the process and its child processes do not gain any additional privileges via suid or sgid bits. This reduces the danger associated with many operations because the possibility of subverting privileged binaries is lessened.

Audit:

You should run the following command:

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
SecurityOpt={{ .HostConfig.SecurityOpt }}'
```

This command should return all the security options currently configured for containers. `no-new-privileges` should be one of them.

Remediation:

You should start your container with the options below:

```
docker run --rm -it --security-opt=no-new-privileges ubuntu bash
```

Impact:

The `no_new_priv` option prevents LSMs like SELinux from allowing processes to acquire new privileges

Default Value:

By default, new privileges are not restricted.

References:

1. https://github.com/projectatomic/atomic-site/issues/269
2. https://github.com/docker/docker/pull/20727
3. https://www.kernel.org/doc/Documentation/prctl/no_new_privs.txt
4. https://lwn.net/Articles/475678/
5. https://lwn.net/Articles/475362/

CIS Controls:

Version 6

5 Controlled Use of Administration Privileges
Controlled Use of Administration Privileges

5.26 Ensure that container health is checked at runtime (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

If the container image does not have an `HEALTHCHECK` instruction defined, you should use the `--health-cmd` parameter at container runtime to check container health.

Rationale:

If the container image you are using does not have a pre-defined `HEALTHCHECK` instruction, use the `--health-cmd` parameter to check container health at runtime.

Based on the reported health status, remedial actions can be taken if necessary.

Audit:

You should run the command below and ensure that all containers are reporting their health status:

```
docker ps --quiet | xargs docker inspect --format '{{ .Id }}: Health={{
.State.Health.Status }}'
```

Remediation:

You should run the container using the `--health-cmd` parameter.
For example:

```
docker run -d --health-cmd='stat /etc/passwd || exit 1' nginx
```

Impact:

None.

Default Value:

By default, health checks are not carried out at container runtime.

References:

1. https://docs.docker.com/engine/reference/run/#healthcheck

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

5.27 Ensure that Docker commands always make use of the latest version of their image (Not Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should always ensure that you are using the latest version of the images within your repository and not cached older versions.

Rationale:

Multiple Docker commands such as `docker pull`, `docker run` etc. are known to have an issue where by default, they extract the local copy of the image, if present, even though there is an updated version of the image with the same tag in the upstream repository. This could lead to using older images containing known vulnerabilites.

Audit:

You should carry out the following steps:
Step 1: Open your image repository and list the image version history for the image you are inspecting.
Step 2: Observe the status when the `docker pull` command is triggered.
If the status is shown as `Image is up to date`, it means that you are getting the cached version of the image.
Step 3: Match the version of the image you are running to the latest version reported in your repository and this will tell you whether you are running the cached version or the latest copy.

Remediation:

You should use proper version pinning mechanisms (the "latest" tag which is assigned by default is still vulnerable to caching attacks) to avoid extracting cached older versions. Version pinning mechanisms should be used for base images, packages, and entire images. You can customize version pinning rules according to your requirements.

Impact:

None

Default Value:

By default, Docker commands extract the local copy unless version pinning mechanisms are used or the local cache is cleared.

References:

1. https://github.com/docker/docker/pull/16609

CIS Controls:

Version 6

18.1 Use Only Vendor-supported Software

For all acquired application software, check that the version you are using is still supported by the vendor. If not, update to the most current version and install all relevant patches and vendor security recommendations.

Version 7

18.3 Verify That Acquired Software is Still Supported

Verify that the version of all software acquired from outside your organization is still supported by the developer or appropriately hardened based on developer security recommendations.

5.28 Ensure that the PIDs cgroup limit is used (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should use the `--pids-limit` flag at container runtime.

Rationale:

Attackers could launch a fork bomb with a single command inside the container. This fork bomb could crash the entire system and would require a restart of the host to make the system functional again. Using the PIDs cgroup parameter `--pids-limit` would prevent this kind of attack by restricting the number of forks that can happen inside a container within a specified time frame.

Audit:

You should run the command below and ensure that `PidsLimit` is not set to 0 or -1. A `PidsLimit` of 0 or -1 means that any number of processes can be forked concurrently inside the container.

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
PidsLimit={{ .HostConfig.PidsLimit }}'
```

Remediation:

Use `--pids-limit` flag with an appropriate value when launching the container.
For example:

```
docker run -it --pids-limit 100 <Image_ID>
```

In the above example, the number of processes allowed to run at any given time is set to 100. After a limit of 100 concurrently running processes is reached, Docker would restrict any new process creation.

Impact:

Set the PIDs limit value as appropriate. Incorrect values might leave containers unusable.

Default Value:

The Default value for `--pids-limit` is `0` which means there is no restriction on the number of forks. Note that the PIDs cgroup limit works only for kernel versions 4.3 and higher.

References:

1. https://github.com/docker/docker/pull/18697
2. https://docs.docker.com/engine/reference/commandline/run/#options

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

5.29 Ensure that Docker's default bridge "docker0" is not used (Not Scored)

Profile Applicability:

- Level 2 - Docker - Linux

Description:

You should not use Docker's default bridge `docker0`. Instead you should use Docker's user-defined networks for container networking.

Rationale:

Docker connects virtual interfaces created in bridge mode to a common bridge called `docker0`. This default networking model is vulnerable to ARP spoofing and MAC flooding attacks as there is no filtering applied to it.

Audit:

You should run the command below, and verify that containers are on a user-defined network and not the default `docker0` bridge.

```
docker network ls --quiet | xargs docker network inspect --format '{{ .Name }}: {{ .Options }}'
```

Remediation:

You should follow the Docker documentation and set up a user-defined network. All the containers should be run in this network.

Impact:

User-defined networks need to be configured and managed in line with organizational security policy.

Default Value:

By default, Docker runs containers within the default `docker0` bridge.

References:

1. https://github.com/nyantec/narwhal
2. https://arxiv.org/pdf/1501.02967
3. https://docs.docker.com/engine/userguide/networking/

CIS Controls:

Version 6

9 Limitation and Control of Network Ports, Protocols, and Services
Limitation and Control of Network Ports, Protocols, and Services

5.30 Ensure that the host's user namespaces are not shared (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should not share the host's user namespaces with containers running on it.

Rationale:

User namespaces ensure that a root process inside the container will be mapped to a non-root process outside the container. Sharing the user namespaces of the host with the container does not therefore isolate users on the host from users in the containers.

Audit:

You should run the command below and ensure that it does not return any value for `UsernsMode`. If it returns a value of `host`, it means that the host user namespace is shared with its containers.

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
UsernsMode={{ .HostConfig.UsernsMode }}'
```

Remediation:

You should not share user namespaces between host and containers.
For example, you should not run the command below:

```
docker run --rm -it --userns=host ubuntu bash
```

Impact:

None

Default Value:

By default, the host user namespace is shared with containers unless user namespace support is enabled.

References:

1. https://docs.docker.com/engine/security/userns-remap/
2. https://docs.docker.com/engine/reference/commandline/run/#options
3. https://github.com/docker/docker/pull/12648

4. https://events.linuxfoundation.org/sites/events/files/slides/User%20Namespaces%20-%20ContainerCon%202015%20-%2016-9-final_0.pdf

CIS Controls:

Version 6

12 Boundary Defense
Boundary Defense

5.31 Ensure that the Docker socket is not mounted inside any containers (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

The Docker socket `docker.sock` should not be mounted inside a container.

Rationale:

If the Docker socket is mounted inside a container it could allow processes running within the container to execute Docker commands which would effectively allow for full control of the host.

Audit:

You should run the following command:

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
Volumes={{ .Mounts }}' | grep docker.sock
```

This would return any instances where `docker.sock` had been mapped to a container as a volume.

Remediation:

You should ensure that no containers mount `docker.sock` as a volume.

Impact:

None

Default Value:

By default, `docker.sock` is not mounted inside containers.

References:

1. https://raesene.github.io/blog/2016/03/06/The-Dangers-Of-Docker.sock/
2. https://forums.docker.com/t/docker-in-docker-vs-mounting-var-run-docker-sock/9450/2
3. https://github.com/docker/docker/issues/21109

CIS Controls:

Version 6

9 Limitation and Control of Network Ports, Protocols, and Services
Limitation and Control of Network Ports, Protocols, and Services

6 Docker Security Operations

This section covers some of the operational security issues associated with Docker deployments. These are best practices that should be followed where possible. Most of the recommendations in this section simply act as reminders that organizations should extend their current security best practices and policies to include containers.

6.1 Ensure that image sprawl is avoided (Not Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

You should not keep a large number of container images on the same host. Use only tagged images as appropriate.

Rationale:

Tagged images are useful if you need to fall back from the "latest" version to a specific version of an image in production. Images with unused or old tags may contain vulnerabilities that might be exploited if instantiated.

Audit:

Step 1 Make a list of all image IDs that are currently instantiated by executing the command below:

```
docker images --quiet | xargs docker inspect --format '{{ .Id }}: Image={{
.Config.Image }}'
```

Step 2: List all the images present on the system by executing the command below:

```
docker images
```

Step 3: Compare the list of image IDs from Step 1 and Step 2 and look for images that are currently not in use. If any unused or old images are found, discuss with the system administrator the need to keep such images on the system. If images are no longer needed they should be deleted.

Remediation:

You should keep only the images that you actually need and establish a workflow to remove old or stale images from the host. Additionally, you should use features such as pull-by-digest to get specific images from the registry.
You can follow the steps below to find unused images on the system so they can be deleted.
Step 1 Make a list of all image IDs that are currently instantiated by executing the command below:

```
docker images --quiet | xargs docker inspect --format '{{ .Id }}: Image={{
.Config.Image }}'
```

Step 2: List all the images present on the system by executing the command below:

```
docker images
```

Step 3: Compare the list of image IDs created from Step 1 and Step 2 to find out images which are currently not being instantiated.
Step 4: Decide if you want to keep the images that are not currently in use. If they are not needed, delete them by executing the following command:

```
docker rmi $IMAGE_ID
```

Alternatively, the `docker system prune` command can be used to remove dangling images which are not tagged or, if necessary, all images that are not currently used by a running container when used with the `-a` option.

Impact:

`docker system prune -a` removes all exited containers as well as all images and volumes that are not referenced by running containers, including for UCP and DTR. This can present problems under three circumstances:

- In Offline environments, where image removal would result in needing to reload the images to the host.
- On DTR hosts, where nodes will not automatically rejoin the DTR cluster and docker system prune -a may remove DTR volumes and erase all DTR state.
- On DTR or UCP hosts where docker system prune -a is simultaneously issued on a majority of UCP managers or DTR nodes whilst the corresponding component UCP or DTR is stopped. This could result in loss of quorum and/or data.

Default Value:

Images and layered filesystems remain accessible on the host until the administrator removes all tags that refer to those images or layers.

References:

1. http://craiccomputing.blogspot.in/2014/09/clean-up-unused-docker-containers-and.html
2. https://forums.docker.com/t/command-to-remove-all-unused-images/20/8
3. https://github.com/docker/docker/issues/9054
4. https://docs.docker.com/engine/reference/commandline/rmi/
5. https://docs.docker.com/engine/reference/commandline/pull/
6. https://github.com/docker/docker/pull/11109

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

6.2 Ensure that container sprawl is avoided (Not Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

You should not keep a large number of containers on the same host.

Rationale:

The flexibility of containers makes it easy to run multiple instances of applications and therefore indirectly leads to Docker images that can exist at varying security patch levels. It also means that you are consuming host resources that otherwise could have been used for running 'useful' containers. Having more than just an essential number of containers on a particular host makes the system vulnerable to mishandling, misconfiguration and fragmentation. You should therefore keep the number of containers on a given host to the minimum number commensurate with serving production applications.

Audit:

Step 1 - Find the total number of containers you have on the host:

```
docker info --format '{{ .Containers }}'
```

Step 2 - Execute the commands below to find the total number of containers that are actually running or in the stopped state on the host.

```
docker info --format '{{ .ContainersStopped }}'
docker info --format '{{ .ContainersRunning }}'
```

If the difference between the number of containers that are stopped on the host and the number of containers that are actually running is excessive, you may be suffering from "Container sprawl" and should review the unused containers for potential deletion.

Remediation:

You should periodically check your container inventory on each host and clean up containers which are not in active use with the command below:

```
docker container prune
```

Impact:

You should retain containers that are actively in use, and delete ones which are no longer needed.

Default Value:

By default, Docker does not restrict the number of containers you may have on a host.

References:

1. https://zeltser.com/security-risks-and-benefits-of-docker-application/
2. http://searchsdn.techtarget.com/feature/Docker-networking-How-Linux-containers-will-change-your-network

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

7 Docker Swarm Configuration

This section lists the recommendations that alter and secure the behavior of the Docker Swarm. If you are not using Docker Swarm then the recommendations in this section do not apply.

7.1 Ensure swarm mode is not Enabled, if not needed (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

Do not enable swarm mode on a Docker engine instance unless this is needed.

Rationale:

By default, a Docker engine instance will not listen on any network ports, with all communications with the client coming over the Unix socket. When Docker swarm mode is enabled on a Docker engine instance, multiple network ports are opened on the system and made available to other systems on the network for the purposes of cluster management and node communications.

Opening network ports on a system increases its attack surface and this should be avoided unless required.

It should be noted that swarm mode is required for the operation of Docker Enterprise components.

Audit:

You should review the output of the `docker info` command. If the output includes `Swarm: active` it indicates that swarm mode has been activated on the Docker engine. In this case, you should confirm if swarm mode on the Docker engine instance is actually needed.

Remediation:

If swarm mode has been enabled on a system in error, you should run the command below:

```
docker swarm leave
```

Impact:

Disabling swarm mode will impact the operation of Docker Enterprise components if these are in use.

Default Value:

By default, Docker swarm mode is not enabled.

References:

1. https://docs.docker.com/engine/reference/commandline/swarm_init/

CIS Controls:

Version 6

9.1 Limit Open Ports, Protocols, and Services
Ensure that only ports, protocols, and services with validated business needs are running on each system.

Version 7

9.2 Ensure Only Approved Ports, Protocols and Services Are Running
Ensure that only network ports, protocols, and services listening on a system with validated business needs, are running on each system.

7.2 Ensure that the minimum number of manager nodes have been created in a swarm (Scored)

Profile Applicability:

- Level 1 - Docker - Linux
- Level 1 - Docker Engine - Enterprise

Description:

You should ensure that the minimum number of required manager nodes is created in a swarm.

Rationale:

Manager nodes within a swarm have control over the swarm and can change its configuration, including modifying security parameters. Having excessive manager nodes could render the swarm more susceptible to compromise.

If fault tolerance is not required in the manager nodes, a single node should be elected as a manger. If fault tolerance is required then the smallest odd number to achieve the appropriate level of tolerance should be configured. This should always be an odd number in order to ensure that a quorum is reached.

Audit:

Run `docker info` and verify the number of managers.

```
docker info --format '{{ .Swarm.Managers }}'
```

Alternatively run the below command.

```
docker node ls | grep 'Leader'
```

Remediation:

If an excessive number of managers is configured, the excess nodes can be demoted to workers using the following command:

```
docker node demote <ID>
```

Where is the node ID value of the manager to be demoted.

Impact:

None

Default Value:

Only a single manager is required to start a given cluster.

References:

1. https://docs.docker.com/engine/swarm/manage-nodes/
2. https://docs.docker.com/engine/swarm/admin_guide/#/add-manager-nodes-for-fault-tolerance

CIS Controls:

Version 6

5 Controlled Use of Administration Privileges
Controlled Use of Administration Privileges

7.3 Ensure that swarm services are bound to a specific host interface (Scored)

Profile Applicability:

- Level 1 - Docker - Linux
- Level 1 - Docker Engine - Enterprise

Description:

By default, Docker swarm services will listen on all interfaces on the host. This may not be necessary for the operation of the swarm where the host has multiple network interfaces.

Rationale:

When a swarm is initialized the default value for the `--listen-addr` flag is `0.0.0.0:2377` which means that swarm services will listen on all interfaces on the host. If a host has multiple network interfaces this may be undesirable as it could expose swarm services to networks which are not involved with the operation of the swarm.

By passing a specific IP address to the `--listen-addr`, a specific network interface can be specified, limiting this exposure.

Audit:

You should check the network listener on port 2377/TCP (the default for docker swarm) and confirm that it is only listening on specific interfaces. For example, in Ubuntu this could be done using the following command:

```
netstat -lt | grep -i 2377
```

Remediation:

Resolving this issues requires re-initialization of the swarm, specifying a specific interface for the `--listen-addr` parameter.

Impact:

None

Default Value:

By default, Docker swarm services listen on all available host interfaces.

References:

1. https://docs.docker.com/engine/reference/commandline/swarm_init/#--listen-addr
2. https://docs.docker.com/engine/swarm/admin_guide/#recover-from-disaster

Notes:

A couple of points I noted looking at this one. there doesn't seem to be a parameter for docker swarm update to change the listen-addr. For the remediation I did wonder if --force-new-swarm could be used to change this, but I'm not sure what other effects that would have on the swarm so just left with a general requirement to re-initialize the swarm.

Also interestingly the node communication service running on 7946/TCP doesn't respect the --listen-addr parameter. this seems like a bug to me, I'll likely file an issue on github for it after a bit more exploration.

CIS Controls:

Version 6

9 Limitation and Control of Network Ports, Protocols, and Services
Limitation and Control of Network Ports, Protocols, and Services

7.4 Ensure that all Docker swarm overlay networks are encrypted (Scored)

Profile Applicability:

- Level 1 - Docker - Linux
- Level 1 - Docker Engine - Enterprise

Description:

Ensure that all Docker swarm overlay networks are encrypted.

Rationale:

By default, data exchanged between containers on nodes on the overlay network is not encrypted. This could potentially expose traffic between containers.

Audit:

You should run the command below to ensure that each overlay network has been encrypted.

```
docker network ls --filter driver=overlay --quiet | xargs docker network
inspect --format '{{.Name}} {{ .Options }}'
```

Remediation:

You should create overlay networks the with `--opt encrypted` flag.

Impact:

None

Default Value:

By default, data exchanged in overlay networks in Docker swarm mode is not encrypted.

References:

1. https://docs.docker.com/engine/userguide/networking/overlay-security-model/
2. https://github.com/docker/docker/issues/24253

CIS Controls:

Version 6

14.2 Encrypt All Sensitive Information Over Less-trusted Networks

All communication of sensitive information over less-trusted networks should be encrypted. Whenever information flows over a network with a lower trust level, the information should be encrypted.

Version 7

14.4 Encrypt All Sensitive Information in Transit

Encrypt all sensitive information in transit.

7.5 Ensure that Docker's secret management commands are used for managing secrets in a swarm cluster (Not Scored)

Profile Applicability:

- Level 2 - Docker - Linux
- Level 2 - Docker Engine - Enterprise

Description:

You should use Docker's in-built secret management command for control of secrets.

Rationale:

Docker has various commands for managing secrets in a swarm cluster.

Audit:

On a swarm manager node, you should run the command below and ensure `docker secret` management is used in your environment where this is in line with your IT security policy.

```
docker secret ls
```

Remediation:

You should follow the `docker secret` documentation and use it to manage secrets effectively.

Impact:

None

Default Value:

Not Applicable

References:

1. https://docs.docker.com/engine/reference/commandline/secret/

CIS Controls:

Version 6

18 Application Software Security

Application Software Security

7.6 Ensure that swarm manager is run in auto-lock mode (Scored)

Profile Applicability:

- Level 2 - Docker - Linux

Description:

You should review whether you wish to run Docker swarm manager in auto-lock mode.

Rationale:

When Docker restarts, both the TLS key used to encrypt communication among swarm nodes, and the key used to encrypt and decrypt Raft logs on disk, are loaded into each manager node's memory. You could protect the mutual TLS encryption key and the key used to encrypt and decrypt Raft logs at rest. This protection could be enabled by initializing the swarm with the `--autolock` flag.

With `--autolock`enabled, when Docker restarts, you must unlock the swarm first, using a key encryption key generated by Docker when the swarm was initialized.

This has benefits in a high security environment, however these should be balanced against the support issues caused by the swarm not starting automatically if, for example the host were to experience an outage.

Audit:

You should run the command below

```
docker info --format 'Swarm Autolock: {{
.Swarm.Cluster.Spec.EncryptionConfig.AutoLockManagers }}'
```

If the result is `true`, auto-lock mode is enable.
You could also run the command below. If a key value is returned, it means that the swarm was initialized with the `--autolock` flag. If the output is `no unlock key is set`, it means that swarm was NOT initialized with the `--autolock` flag. This should be reviewed in line with the organization's IT Security policy.

```
docker swarm unlock-key
```

Remediation:

If you are initializing a swarm, use the command below.

```
docker swarm init --autolock
```

If you want to set `--autolock` on an existing swarm manager node, use the following command.

```
docker swarm update --autolock
```

Impact:

A swarm in auto-lock mode will not recover from a restart without manual intervention from an administrator to enter the unlock key. This may not always be desirable, and should be reviewed at a policy level.

Default Value:

By default, the swarm manager does not run in auto-lock mode.

References:

1. https://docs.docker.com/engine/swarm/swarm_manager_locking/

CIS Controls:

Version 6

14.2 Encrypt All Sensitive Information Over Less-trusted Networks
All communication of sensitive information over less-trusted networks should be encrypted. Whenever information flows over a network with a lower trust level, the information should be encrypted.

Version 7

14.4 Encrypt All Sensitive Information in Transit
Encrypt all sensitive information in transit.

7.7 Ensure that the swarm manager auto-lock key is rotated periodically (Not Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

You should rotate the swarm manager auto-lock key periodically.

Rationale:

The swarm manager auto-lock key is not automatically rotated. Good security practice is to rotate keys.

Audit:

Currently, there is no mechanism to find out when the key was last rotated on a swarm manager node. You should check with the system administrator to see if there is a key rotation process, and how often the key is rotated.

Remediation:

You should run the command below to rotate the keys.

```
docker swarm unlock-key --rotate
```

Additionally, to facilitate auditing of this recommendation, you should maintain key rotation records and ensure that you establish a pre-defined frequency for key rotation.

Impact:

None

Default Value:

By default, keys are not rotated automatically.

References:

1. https://docs.docker.com/engine/reference/commandline/swarm_unlock-key/

CIS Controls:

Version 6

14.2 Encrypt All Sensitive Information Over Less-trusted Networks

All communication of sensitive information over less-trusted networks should be encrypted. Whenever information flows over a network with a lower trust level, the information should be encrypted.

Version 7

14.4 Encrypt All Sensitive Information in Transit

Encrypt all sensitive information in transit.

7.8 Ensure that node certificates are rotated as appropriate (Not Scored)

Profile Applicability:

- Level 2 - Docker - Linux
- Level 2 - Docker Engine - Enterprise

Description:

You should rotate swarm node certificates in line with your organizational security policy.

Rationale:

Docker Swarm uses TLS for clustering operations between its nodes. Certificate rotation ensures that in an event such as a compromised node or key, it is difficult to impersonate a node. By default, node certificates are rotated every 90 days, but you should rotate them more often or as appropriate in your environment.

Audit:

Run one of the commands below and ensure that the node certificate `Expiry Duration` is set as appropriate.

```
docker info | grep "Expiry Duration"

docker info --format 'NodeCertExpiry: {{
.Swarm.Cluster.Spec.CAConfig.NodeCertExpiry }}'
```

Remediation:

You should run the command to set the desired expiry time on the node certificate.
For example:

```
docker swarm update --cert-expiry 48h
```

Impact:

None

Default Value:

By default, node certificates are rotated automatically every 90 days.

References:

1. https://docs.docker.com/engine/reference/commandline/swarm_update/#examples

CIS Controls:

Version 6

14.2 Encrypt All Sensitive Information Over Less-trusted Networks
All communication of sensitive information over less-trusted networks should be encrypted. Whenever information flows over a network with a lower trust level, the information should be encrypted.

Version 7

14.4 Encrypt All Sensitive Information in Transit
Encrypt all sensitive information in transit.

7.9 Ensure that CA certificates are rotated as appropriate (Not Scored)

Profile Applicability:

- Level 2 - Docker - Linux

Description:

You should rotate root CA certificates as appropriate.

Rationale:

Docker Swarm uses TLS for clustering operations between its nodes. Certificate rotation ensures that in an event such as a compromised node or key, it is difficult to impersonate a node. Node certificates depend upon root CA certificates. For operational security, it is important to rotate these frequently. Currently, root CA certificates are not rotated automatically and you should therefore establish a process for rotating them in line with your organizational security policy.

Audit:

You should check the time stamp on the root CA certificate file.
For example:

```
ls -l /var/lib/docker/swarm/certificates/swarm-root-ca.crt
```

The certificate should show a time stamp in line with the organizational rotation policy.

Remediation:

You should run the command below to rotate a certificate.

```
docker swarm ca --rotate
```

Impact:

None

Default Value:

By default, root CA certificates are not rotated.

References:

1. https://docs.docker.com/engine/swarm/how-swarm-mode-works/pki/#rotating-the-ca-certificate

CIS Controls:

Version 6

14.2 Encrypt All Sensitive Information Over Less-trusted Networks
All communication of sensitive information over less-trusted networks should be encrypted. Whenever information flows over a network with a lower trust level, the information should be encrypted.

Version 7

14.4 Encrypt All Sensitive Information in Transit
Encrypt all sensitive information in transit.

7.10 Ensure that management plane traffic is separated from data plane traffic (Not Scored)

Profile Applicability:

- Level 2 - Docker - Linux
- Level 2 - Docker Engine - Enterprise

Description:

You should separate management plane traffic from data plane traffic.

Rationale:

Separating management plane traffic from data plane traffic ensures that these types of traffic are segregated from each other. These traffic flows can then be individually monitored and tied to different traffic control policies and monitoring. This also ensures that the management plane is always reachable even if there is a great deal of traffic on the data plane.

Audit:

You should run the command below on each swarm node and ensure that the management plane address is different from the data plane address.

```
docker node inspect  --format '{{ .Status.Addr }}' self
```

Remediation:

You should initialize the swarm with dedicated interfaces for management and data planes respectively.
For example,

```
docker swarm init --advertise-addr=192.168.0.1 --data-path-addr=17.1.0.3
```

Impact:

This requires two network interfaces per node.

Default Value:

By default, data plane traffic is not separated from management plane traffic.

References:

1. https://docs.docker.com/engine/reference/commandline/swarm_init/#--data-path-addr
2. https://github.com/moby/moby/issues/33938
3. https://github.com/moby/moby/pull/32717

CIS Controls:

Version 6

18 Application Software Security
Application Software Security

8 Docker Enterprise Configuration

This section contains recommendations for securing Docker Enterprise components.

8.1 Universal Control Plane Configuration

This section contains recommendations for securing the Universal Control Plane Configuration in Docker Enterprise Edition.

8.1.1 Configure the LDAP authentication service (Scored)

Profile Applicability:

- Level 1 - Docker Engine - Enterprise

Description:

By default, the Universal Control Plane is configured to use the managed user authentication service. UCP should instead be configured to use one or more external LDAP endpoints for authenticating users as this can enable more granular control over authentication and authorization.

Rationale:

UCP's built-in managed user authentication system only supports user creation, deletion and disablement. By using an external LDAP endpoint, you can have more control over the users, groups and other hierarchical organizations that can access and manipulate resources via UCP.

Audit:

The UCP "Admin Settings" UI can be used to validate that LDAP integration has been enabled. Instructions for doing so can be found here.
The Docker CLI can also be used to audit the LDAP integration. From a UCP "manager" node, you can execute the following commands to get the current configuration:

```
CURRENT_CONFIG_NAME=$(docker service inspect --format '{{ range $config :=
.Spec.TaskTemplate.ContainerSpec.Configs }}{{ $config.ConfigName }}{{ "\n"
}}{{ end }}' ucp-agent | grep 'com.docker.ucp.config-')
docker config inspect --format '{{ printf "%s" .Spec.Data }}'
$CURRENT_CONFIG_NAME | grep backend
```

The result should be `backend = "ldap"`.

Remediation:

You can configure LDAP integration via the UCP "Admin Settings" UI by following the instructions here. LDAP integration can also be enabled via a configuration file by following the instructions here.

Impact:

None.

Default Value:

By default, the built-in managed user database is enabled.

References:

1. https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/external-auth/
2. https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/external-auth/enable-ldap-config-file/
3. http://success.docker.com/article/Docker_Reference_Architecture-_Securing_Docker_EE_and_Security_Best_Practices

8.1.2 Use external certificates (Scored)

Profile Applicability:

- Level 1 - Docker Engine - Enterprise

Description:

When you install the Universal Control Plane without providing your own TLS certificates, it will, by default, configure self-signed certificates. You should instead use certificates signed by an external, trusted certified authority as these mitigate the overhead of having to distribute certificate authority certificates to all of the nodes in a Universal Control Plane cluster and additionally are in line with good security practice.

Rationale:

By default, UCP is configured to use untrusted, self-signed certificates. Using UCP with externally trusted certificate authorities is a more streamlined and secure option.

Audit:

You can use the `openssl` utility to validate your UCP cluster's certificate chain as follows:

```
openssl s_client -showcerts -connect <UCP_FQDN:443>
```

The result should be indicative of your externally-signed certificate chain.

Remediation:

You can configure your own certificates for UCP either during installation or after installation via the UCP "Admin Settings" user interface.
Customize certificates during installation:

1. Create a volume named `ucp-controller-server-certs` on your primary UCP Manager installation node:

```
docker volume create ucp-controller-server-certs
```

2. Copy your external certificate authority's public certificate file (ca.pem) and your signed certificate (cert.pem) and key (key.pem) files to the root directory of the volume

```
cp ca.pem cert.pem key.pem $(docker volume inspect --format '{{ .Mountpoint
}}' ucp-controller-server-certs)/
```

3. Run the UCP installation command with the `--external-server-cert` flag

Customize certificates post-installation via the "Admin Settings" UI:
Refer to the instructions at https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/use-your-own-tls-certificates/#configure-ucp-to-use-your-own-tls-certificates-and-keys for configuring your own certificates via the UCP UI.

Impact:

None.

Default Value:

Self-signed certificates are configured by default.

References:

1. https://docs.docker.com/datacenter/ucp/2.2/guides/admin/install/plan-installation/#use-an-external-certificate-authority
2. https://docs.docker.com/datacenter/ucp/2.2/reference/cli/install/#description
3. https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/use-your-own-tls-certificates/

8.1.3 Enforce the use of client certificate bundles for unprivileged users (Not Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

While you can communicate with a UCP cluster by connecting as a user with administrative permissions directly to one of the UCP Manager nodes, we recommend that unprivileged users should instead be provided with client certificate bundles so that their access rights are controlled via the built-in role-based access control (RBAC) model.

Rationale:

UCP cluster administrators can leverage the built-in RBAC capabilities within Docker and provide client certificate bundles to unprivileged users rather than allowing them to connect directly to the Manager and/or Worker nodes in the cluster. This prevents unprivileged users from being able to directly access or manipulate cluster resources. With the use of UCP client certificate bundles you do not need to include standard users in the "docker" security group and instead you can facilitate user access to the cluster via RBAC.

Audit:

UCP cluster administrators can audit client certificate bundles on a per-user basis. You can verify that a user's client certificate bundle has been created by navigating to the USER MANAGEMENT | USERS interface in UCP, selecting the user from the list, clicking on the "Configure" button from the right-hand navigation menu, and selecting "Client Bundle" from the drop-down. From there, a list of client bundles assigned to the user will appear. This page also allows administrators to revoke client certificate bundles when necessary.

Remediation:

Client certificate bundles can be created in one of two ways:
User Management UI
UCP Administrators can provision client certificate bundles on behalf of users by navigating to the USER MANAGEMENT | USERS interface in UCP, selecting the user from the list, clicking on the "Configure" button from the right-hand navigation, and selecting "Client Bundle" from the drop-down. The "New Client Bundle" link can be selected to create a client bundle. This will trigger a download of the bundle as a .zip file.
Self-Provision

Users with access to the UCP console can create client certificate bundles themselves. After logging into the console, the user can select their username drop-down from the top-left corner of the navigation page and select the "My Profile" option. From there, the "New Client Bundle" link can be selected to create a client bundle and this will trigger a download of the bundle as a .zip file.

Impact:

None.

References:

1. https://docs.docker.com/ee/ucp/user-access/cli/
2. https://success.docker.com/article/Docker_Reference_Architecture-_Securing_Docker_EE_and_Security_Best_Practices#ucpsecurity
3. https://docs.docker.com/ee/ucp/authorization/

8.1.4 Configure applicable cluster role-based access control policies (Not Scored)

Profile Applicability:

- Level 1 - Docker - Linux
- Level 1 - Docker Engine - Enterprise

Description:

The Universal Control Plane provides robust role-based access control (RBAC) capabilities that can be used to further harden a deployment. Building off of the default set of RBAC components which includes subjects, roles, resource collections, and grants, an appropriate RBAC model should be developed that aligns with your organization's IT Security policies. This involves creating custom roles and collections.

Rationale:

The RBAC functionality provided by UCP includes a set of defaults that should be customized to satisfy your organization's security requirements. The following roles are included by default: **None**, **View Only**, **Restricted Control**, **Scheduler**, and **Full Control**. While by default, these roles are applicable to a number of simple management and application deployment scenarios, they are too broad in regards to the permissions allocated by each.

As such, custom roles should be created to extend these defaults.

Audit:

The UCP "User Management" UI can be used to validate that the configured RBAC model satisfies the requirements of your organization.

Remediation:

UCP RBAC components can be configured as required via the UCP "User Management" UI.

Impact:

None

References:

1. https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/

2. https://success.docker.com/article/Docker_Reference_Architecture-_Securing_Docker_EE_and_Security_Best_Practices#ucpsecurity

8.1.5 Enable signed image enforcement (Scored)

Profile Applicability:

- Level 1 - Docker - Linux
- Level 2 - Docker Engine - Enterprise

Description:

The Universal Control Plane includes the ability to enforce running of only images that have been signed by members of a particular group. This capability should be enabled to prevent unsigned images from being deployed to your cluster.

Rationale:

Running untrusted containers poses a risk to the operation of your Docker platform. Combined with the Docker Content Trust recommendations in Section 4, signed image enforcement in UCP gives you more control over the validity and origination of your Docker images prior to deployment. Signed image enforcement can prohibit images that are unsigned, have malformed signatures, and/or compromised signatures from being deployed.

Audit:

The "Docker Content Trust" page under the UCP "Admin Settings" UI can be used to verify that this setting has been enabled.

Remediation:

References:

1. https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/run-only-the-images-you-trust/
2. https://success.docker.com/article/Docker_Reference_Architecture-_Securing_Docker_EE_and_Security_Best_Practices

8.1.6 Set the Per-User Session Limit to a value of '3' or lower (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

The "Per User Limit" Login Session Control which is configured in the UCP "Admin Settings" | "Authentication & Authorization" section specifies the maximum number of sessions that any user can have active at any given time. If creating a new session would put a user over this limit then the least recently used session will be deleted. Set this limit to a lower value but greater than '0' to prevent users from initiating an unecessarily high number of concurrent sessions. This limit applies to users that are authenticated to UCP and/or DTR as the built-in authentication and authorization backplane in UCP serves both UCP and DTR.

Rationale:

By default, UCP sets the "Per User Limit" value to '10' which may be too high for the number of concurrent sessions that users are allotted. Users who are able to maintain a large number of concurrent sessions could be subject to phishing attacks or similar that result in unauthorized sessions with a UCP and/or DTR cluster. Furthermore, setting a value of '0' disables limiting the number of sessions that users may have, and this is not in line with good security practice.

Audit:

As a Docker Enterprise Administrator, you should execute the following commands from a machine with connectivity to the UCP management console. Replace `[ucp_url]` with your UCP URL, `[ucp_username]` with the username of a Docker Enterprise Administrator and `[ucp_password]` with the password of a Docker Enterprise Administrator.
Step 1: Retrieve a UCP API token
Linux (requires curl and jq):

```
$ AUTHTOKEN=$(curl -sk -d
'{"username":"[ucp_username]","password":"[ucp_password]"}'
https://[ucp_url]/auth/login | jq -r .auth_token)
```

Step 2: Retrieve UCP config
Linux (requires curl):

```
$ curl -sk -H "Authorization: Bearer $AUTHTOKEN"
https://[ucp_url]/api/ucp/config-toml
```

Step 3: Look for the `per_user_limit` entry under the `[auth.sessions]` section in the output, and verify that it is set to a value of '3' or lower, but greater than '0'

Remediation:

As a Docker Enterprise Administrator, execute the following commands from a machine with connectivity to the UCP management console. Replace `[ucp_url]` with your UCP URL, `[ucp_username]` with the username of a Docker Enterprise Administrator and `[ucp_password]` with the password of a Docker Enterprise Administrator.
Step 1: Retrieve a UCP API token
Linux (requires curl and jq):

```
$ AUTHTOKEN=$(curl -sk -d
'{"username":"[ucp_username]","password":"[ucp_password]"}'
https://[ucp_url]/auth/login | jq -r .auth_token)
```

Step 2: Retrieve and save UCP config
Linux (requires curl):

```
$ curl -sk -H "Authorization: Bearer $AUTHTOKEN"
https://[ucp_url]/api/ucp/config-toml > ucp-config.toml
```

Step 3: Open the `ucp-config.toml` file, set the `per_user_limit` entry under the `[auth.sessions]` section to a value of '3' or lower, but greater than '0'. Save the file.
Step 4: Execute the following command to update UCP with the new configuration:
Linux (requires curl):

```
$ curl -sk -H "Authorization: Bearer $AUTHTOKEN" --upload-file ucp-
config.toml https://[ucp_url]/api/ucp/config-toml
```

Impact:

None.

Default Value:

The "Per User Limit" Login Session Control is set to a value of '10' by default.

References:

1. https://docs.docker.com/ee/ucp/admin/configure/set-session-timeout/

CIS Controls:

Version 7

16 Account Monitoring and Control

Account Monitoring and Control

8.1.7 Set the "Lifetime Minutes" and "Renewal Threshold Minutes" values to '15' or lower and '0' respectively (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

The "Lifetime Minutes" Login Session Control which is configured in the UCP "Admin Settings" | "Authentication & Authorization" section specifies the initial lifetime (in minutes) of a session from the moment it is generated. This should be set to a value of '15' or lower so as to restrict a Docker Enterprise user's session length to 15 minutes or less. The "Renewal Threshold Minutes" Login Session Control which is also configured in the UCP "Admin Settings" | "Authentication & Authorization" section indicates the period of time (in minutes) before the expiration of a session where, if set, a session will be extended by the current configured lifetime from then. This value cannot be greater than the configured lifetime. A value equal to the lifetime means that sessions will be extended with every use. A value of zero indicates that sessions should never be extended, but this may result in unexpectedly being logged out if the session expires while performing a series of actions in the UI. This value should be set to '0' to prevent a user's session from being extended for any period of time.

Rationale:

By default, the values of "Lifetime Minutes" and "Renewal Threshold Minutes" are set to '60' and '20' respectively. These values are too high for some organizations and could result in users maintaining active sessions to a Docker Enterprise cluster for a longer period of time than is desired. This makes users prone to session compromise if they are away from their workstations for an extended period of time.

Audit:

As a Docker Enterprise Administrator, execute the following commands from a machine with connectivity to the UCP management console. Replace `[ucp_url]` with your UCP URL, `[ucp_username]` with the username of a Docker Enterprise Administrator and `[ucp_password]` with the password of a Docker Enterprise Administrator.
Step 1: Retrieve a UCP API token
Linux (requires curl and jq):

```
$ AUTHTOKEN=$(curl -sk -d
'{"username":"[ucp_username]","password":"[ucp_password]"}'
https://[ucp_url]/auth/login | jq -r .auth_token)
```

Step 2: Retrieve UCP config
Linux (requires curl):

```
$ curl -sk -H "Authorization: Bearer $AUTHTOKEN"
https://[ucp_url]/api/ucp/config-toml
```

Step 3: Look for the `lifetime_minutes` and `renewal_threshold_minutes` entries under the `[auth.sessions]` section in the output, and verify that they are set to values of '15' or lower and '0' respectively

Remediation:

As a Docker Enterprise Administrator, execute the following commands from a machine with connectivity to the UCP management console. Replace `[ucp_url]` with your UCP URL, `[ucp_username]` with the username of a Docker Enterprise Administrator and `[ucp_password]` with the password of a Docker Enterprise Administrator.
Step 1: Retrieve a UCP API token
Linux (requires curl and jq):

```
$ AUTHTOKEN=$(curl -sk -d
'{"username":"[ucp_username]","password":"[ucp_password]"}'
https://[ucp_url]/auth/login | jq -r .auth_token)
```

Step 2: Retrieve and save UCP config
Linux (requires curl):

```
$ curl -sk -H "Authorization: Bearer $AUTHTOKEN"
https://[ucp_url]/api/ucp/config-toml > ucp-config.toml
```

Step 3: Open the `ucp-config.toml` file, set the `lifetime_minutes` and `renewal_threshold_minutes` entries under the `[auth.sessions]` section to values of '15' or lower and '0' respectively. Save the file.
Step 4: Execute the following command to update UCP with the new configuration:
Linux (requires curl):

```
$ curl -sk -H "Authorization: Bearer $AUTHTOKEN" --upload-file ucp-
config.toml https://[ucp_url]/api/ucp/config-toml
```

Impact:

Setting the "Lifetime Minutes" setting to a value that is too lower would result in users having to constantly re-authenticate to their Docker Enterprise cluster.

Default Value:

By default, the values of "Lifetime Minutes" and "Renewal Threshold Minutes" are set to '60' and '20' respectively.

References:

1. https://docs.docker.com/ee/ucp/admin/configure/set-session-timeout/

CIS Controls:

Version 7

16 Account Monitoring and Control
Account Monitoring and Control

8.2 Docker Trusted Registry Configuration

This section contains recommendations for securing the Docker Trusted Registry Configuration in Docker Enterprise Edition.

8.2.1 Enable image vulnerability scanning (Scored)

Profile Applicability:

- Level 1 - Docker - Linux

Description:

It is important to ensure that your Docker images are free from vulnerabilities. The Docker Trusted Registry (DTR) includes image vulnerability scanning which can check any packages included in your image against known vulnerability databases. This capability should be enabled to satisfy this recommendation.

Rationale:

Running Docker containers based on images with known vulnerabilities exposes your organization to a greater level of risk. The vulnerability scanning service included with DTR can check the signature of any packages included in your image's layers against both the MITRE Common Vulnerabilities and Exposures (CVE) database and NIST National Vulnerability Database (NVD). Docker Inc. maintains a security scanning database which is an aggregation of the MITRE CVE and NIST NVD data that can be read by DTR. DTR's vulnerability scanning capabilty can operate in online mode, where it connects directly to Docker's database at https://dss-cve-updates.docker.com/. It can also operate in offline mode, where the user must download a .tar file that contains the aggregated database that DTR can read. Docker Inc. updates this database on a daily basis.

Audit:

The "Security" tab in the DTR "Settings" UI can be used to verify that image scanning has been enabled, along with information regarding the synchronization status of the vulnerability database. The Image Scanning configuration can also be retrieved by executing an HTTP GET request to the DTR API using the cURL command as follows:

```
$ curl -X GET "https://<YOUR_DTR_URL>/api/v0/imagescan/status" -H "accept:
application/json"
```

Remediation:

You can navigate to DTR "Settings" UI and select the "Security" tab to access the image scanning configuration. Select the "Enable Scanning" slider to enable this functionality. You can also enable the Image Scanning capability via the DTR API using the cURL command as follows:

```
$ curl -X POST "https://<YOUR_DTR_URL>/api/v0/meta/settings" -H "accept:
application/json" -H "content-type: application/json" -d "{
\"scanningEnabled\": true}"
```

Impact:

None.

Default Value:

The image scanning feature is disabled by default.

References:

1. https://docs.docker.com/ee/dtr/admin/configure/set-up-vulnerability-scans/
2. https://success.docker.com/article/Docker_Reference_Architecture-_Securing_Docker_EE_and_Security_Best_Practices#dtrsecurity

Appendix: Summary Table

Control		Set Correctly	
		Yes	No
1	**Host Configuration**		
1.1	**General Configuration**		
1.1.1	Ensure the container host has been Hardened (Not Scored)	☐	☐
1.1.2	Ensure that the version of Docker is up to date (Not Scored)	☐	☐
1.2	**Linux Hosts Specific Configuration**		
1.2.1	Ensure a separate partition for containers has been created (Scored)	☐	☐
1.2.2	Ensure only trusted users are allowed to control Docker daemon (Scored)	☐	☐
1.2.3	Ensure auditing is configured for the Docker daemon (Scored)	☐	☐
1.2.4	Ensure auditing is configured for Docker files and directories - /var/lib/docker (Scored)	☐	☐
1.2.5	Ensure auditing is configured for Docker files and directories - /etc/docker (Scored)	☐	☐
1.2.6	Ensure auditing is configured for Docker files and directories - docker.service (Scored)	☐	☐
1.2.7	Ensure auditing is configured for Docker files and directories - docker.socket (Scored)	☐	☐
1.2.8	Ensure auditing is configured for Docker files and directories - /etc/default/docker (Scored)	☐	☐
1.2.9	Ensure auditing is configured for Docker files and directories - /etc/sysconfig/docker (Scored)	☐	☐
1.2.10	Ensure auditing is configured for Docker files and directories - /etc/docker/daemon.json (Scored)	☐	☐
1.2.11	Ensure auditing is configured for Docker files and directories - /usr/bin/containerd (Scored)	☐	☐
1.2.12	Ensure auditing is configured for Docker files and directories - /usr/sbin/runc (Scored)	☐	☐
2	**Docker daemon configuration**		
2.1	Ensure network traffic is restricted between containers on the default bridge (Scored)	☐	☐
2.2	Ensure the logging level is set to 'info' (Scored)	☐	☐
2.3	Ensure Docker is allowed to make changes to iptables (Scored)	☐	☐
2.4	Ensure insecure registries are not used (Scored)	☐	☐
2.5	Ensure aufs storage driver is not used (Scored)	☐	☐
2.6	Ensure TLS authentication for Docker daemon is configured	☐	☐

	(Scored)		
2.7	Ensure the default ulimit is configured appropriately (Not Scored)	☐	☐
2.8	Enable user namespace support (Scored)	☐	☐
2.9	Ensure the default cgroup usage has been confirmed (Scored)	☐	☐
2.10	Ensure base device size is not changed until needed (Scored)	☐	☐
2.11	Ensure that authorization for Docker client commands is enabled (Scored)	☐	☐
2.12	Ensure centralized and remote logging is configured (Scored)	☐	☐
2.13	Ensure live restore is enabled (Scored)	☐	☐
2.14	Ensure Userland Proxy is Disabled (Scored)	☐	☐
2.15	Ensure that a daemon-wide custom seccomp profile is applied if appropriate (Not Scored)	☐	☐
2.16	Ensure that experimental features are not implemented in production (Scored)	☐	☐
2.17	Ensure containers are restricted from acquiring new privileges (Scored)	☐	☐
3	**Docker daemon configuration files**		
3.1	Ensure that the docker.service file ownership is set to root:root (Scored)	☐	☐
3.2	Ensure that docker.service file permissions are appropriately set (Scored)	☐	☐
3.3	Ensure that docker.socket file ownership is set to root:root (Scored)	☐	☐
3.4	Ensure that docker.socket file permissions are set to 644 or more restrictive (Scored)	☐	☐
3.5	Ensure that the /etc/docker directory ownership is set to root:root (Scored)	☐	☐
3.6	Ensure that /etc/docker directory permissions are set to 755 or more restrictively (Scored)	☐	☐
3.7	Ensure that registry certificate file ownership is set to root:root (Scored)	☐	☐
3.8	Ensure that registry certificate file permissions are set to 444 or more restrictively (Scored)	☐	☐
3.9	Ensure that TLS CA certificate file ownership is set to root:root (Scored)	☐	☐
3.10	Ensure that TLS CA certificate file permissions are set to 444 or more restrictively (Scored)	☐	☐
3.11	Ensure that Docker server certificate file ownership is set to root:root (Scored)	☐	☐
3.12	Ensure that the Docker server certificate file permissions are set to 444 or more restrictively (Scored)	☐	☐
3.13	Ensure that the Docker server certificate key file ownership is set to root:root (Scored)	☐	☐

3.14	Ensure that the Docker server certificate key file permissions are set to 400 (Scored)	☐	☐
3.15	Ensure that the Docker socket file ownership is set to root:docker (Scored)	☐	☐
3.16	Ensure that the Docker socket file permissions are set to 660 or more restrictively (Scored)	☐	☐
3.17	Ensure that the daemon.json file ownership is set to root:root (Scored)	☐	☐
3.18	Ensure that daemon.json file permissions are set to 644 or more restrictive (Scored)	☐	☐
3.19	Ensure that the /etc/default/docker file ownership is set to root:root (Scored)	☐	☐
3.20	Ensure that the /etc/sysconfig/docker file ownership is set to root:root (Scored)	☐	☐
3.21	Ensure that the /etc/sysconfig/docker file permissions are set to 644 or more restrictively (Scored)	☐	☐
3.22	Ensure that the /etc/default/docker file permissions are set to 644 or more restrictively (Scored)	☐	☐
4	**Container Images and Build File Configuration**		
4.1	Ensure that a user for the container has been created (Scored)	☐	☐
4.2	Ensure that containers use only trusted base images (Not Scored)	☐	☐
4.3	Ensure that unnecessary packages are not installed in the container (Not Scored)	☐	☐
4.4	Ensure images are scanned and rebuilt to include security patches (Not Scored)	☐	☐
4.5	Ensure Content trust for Docker is Enabled (Scored)	☐	☐
4.6	Ensure that HEALTHCHECK instructions have been added to container images (Scored)	☐	☐
4.7	Ensure update instructions are not use alone in the Dockerfile (Not Scored)	☐	☐
4.8	Ensure setuid and setgid permissions are removed (Not Scored)	☐	☐
4.9	Ensure that COPY is used instead of ADD in Dockerfiles (Not Scored)	☐	☐
4.10	Ensure secrets are not stored in Dockerfiles (Not Scored)	☐	☐
4.11	Ensure only verified packages are are installed (Not Scored)	☐	☐
5	**Container Runtime Configuration**		
5.1	Ensure that, if applicable, an AppArmor Profile is enabled (Scored)	☐	☐
5.2	Ensure that, if applicable, SELinux security options are set (Scored)	☐	☐
5.3	Ensure that Linux kernel capabilities are restricted within containers (Scored)	☐	☐

5.4	Ensure that privileged containers are not used (Scored)	☐	☐
5.5	Ensure sensitive host system directories are not mounted on containers (Scored)	☐	☐
5.6	Ensure sshd is not run within containers (Scored)	☐	☐
5.7	Ensure privileged ports are not mapped within containers (Scored)	☐	☐
5.8	Ensure that only needed ports are open on the container (Not Scored)	☐	☐
5.9	Ensure that the host's network namespace is not shared (Scored)	☐	☐
5.10	Ensure that the memory usage for containers is limited (Scored)	☐	☐
5.11	Ensure that CPU priority is set appropriately on containers (Scored)	☐	☐
5.12	Ensure that the container's root filesystem is mounted as read only (Scored)	☐	☐
5.13	Ensure that incoming container traffic is bound to a specific host interface (Scored)	☐	☐
5.14	Ensure that the 'on-failure' container restart policy is set to '5' (Scored)	☐	☐
5.15	Ensure that the host's process namespace is not shared (Scored)	☐	☐
5.16	Ensure that the host's IPC namespace is not shared (Scored)	☐	☐
5.17	Ensure that host devices are not directly exposed to containers (Not Scored)	☐	☐
5.18	Ensure that the default ulimit is overwritten at runtime if needed (Not Scored)	☐	☐
5.19	Ensure mount propagation mode is not set to shared (Scored)	☐	☐
5.20	Ensure that the host's UTS namespace is not shared (Scored)	☐	☐
5.21	Ensure the default seccomp profile is not Disabled (Scored)	☐	☐
5.22	Ensure that docker exec commands are not used with the privileged option (Scored)	☐	☐
5.23	Ensure that docker exec commands are not used with the user=root option (Not Scored)	☐	☐
5.24	Ensure that cgroup usage is confirmed (Scored)	☐	☐
5.25	Ensure that the container is restricted from acquiring additional privileges (Scored)	☐	☐
5.26	Ensure that container health is checked at runtime (Scored)	☐	☐
5.27	Ensure that Docker commands always make use of the latest version of their image (Not Scored)	☐	☐
5.28	Ensure that the PIDs cgroup limit is used (Scored)	☐	☐
5.29	Ensure that Docker's default bridge "docker0" is not used (Not Scored)	☐	☐
5.30	Ensure that the host's user namespaces are not shared	☐	☐

	(Scored)		
5.31	Ensure that the Docker socket is not mounted inside any containers (Scored)	☐	☐
6	**Docker Security Operations**		
6.1	Ensure that image sprawl is avoided (Not Scored)	☐	☐
6.2	Ensure that container sprawl is avoided (Not Scored)	☐	☐
7	**Docker Swarm Configuration**		
7.1	Ensure swarm mode is not Enabled, if not needed (Scored)	☐	☐
7.2	Ensure that the minimum number of manager nodes have been created in a swarm (Scored)	☐	☐
7.3	Ensure that swarm services are bound to a specific host interface (Scored)	☐	☐
7.4	Ensure that all Docker swarm overlay networks are encrypted (Scored)	☐	☐
7.5	Ensure that Docker's secret management commands are used for managing secrets in a swarm cluster (Not Scored)	☐	☐
7.6	Ensure that swarm manager is run in auto-lock mode (Scored)	☐	☐
7.7	Ensure that the swarm manager auto-lock key is rotated periodically (Not Scored)	☐	☐
7.8	Ensure that node certificates are rotated as appropriate (Not Scored)	☐	☐
7.9	Ensure that CA certificates are rotated as appropriate (Not Scored)	☐	☐
7.10	Ensure that management plane traffic is separated from data plane traffic (Not Scored)	☐	☐
8	**Docker Enterprise Configuration**		
8.1	**Universal Control Plane Configuration**		
8.1.1	Configure the LDAP authentication service (Scored)	☐	☐
8.1.2	Use external certificates (Scored)	☐	☐
8.1.3	Enforce the use of client certificate bundles for unprivileged users (Not Scored)	☐	☐
8.1.4	Configure applicable cluster role-based access control policies (Not Scored)	☐	☐
8.1.5	Enable signed image enforcement (Scored)	☐	☐
8.1.6	Set the Per-User Session Limit to a value of '3' or lower (Scored)	☐	☐
8.1.7	Set the "Lifetime Minutes" and "Renewal Threshold Minutes" values to '15' or lower and '0' respectively (Scored)	☐	☐
8.2	**Docker Trusted Registry Configuration**		
8.2.1	Enable image vulnerability scanning (Scored)	☐	☐

Appendix: Change History

Date	Version	Changes for this version
01-19-17	1.0.0	Initial Release
07-06-17	1.1.0	NEW - Ensure containers are restricted from acquiring new privileges. Ticket #5222
07-06-17	1.1.0	NEW - Ensure node certificates are rotated as appropriate. Ticket #5227
07-06-17	1.1.0	NEW - Ensure CA certificates are rotated as appropriate. Ticket #5228
07-06-17	1.1.0	NEW - Ensure management plane traffic has been separated from data plane traffic. Ticket # 5229
07-06-17	1.1.0	MODIFY - clarified the intent of the recommendation - Ensure network traffic is restricted between containers on the default bridge. Ticket # 5210
07-06-17	1.1.0	DELETED – 6.1 - Perform regular security audits of your host system and containers. Ticket # 5223
07-06-17	1.1.0	DELETED – 6.2 - Monitor Docker containers usage, performance and metering. Ticket # 5224
07-06-17	1.1.0	DELETED – 6.3 -Backup container data. Ticket # 5225
07-06-17	1.1.0	UPDATES - Updated several reference URLs

07-06-17	1.1.0	NEW - New Section - "7 Docker Swarm Configuration"
07-06-17	1.1.0	NEW – CIS CONTROLS Mappings
07-06-17	1.1.0	UPDATES – Updated all recommendation titles to conform to CIS standard.
07-06-17	1.1.0	UPDATES – Cleaned up formatting of benchmark
07-15-19	1.2.0	UPDATE - Change section header and summary to read "Docker Enterprise Configuration" Ticket #7163
07-15-19	1.2.0	UPDATE - Ensure that Linux kernel capabilities are restricted within containers Ticket #6899
07-15-19	1.2.0	UPDATE - Ensure that swarm services are bound to a specific host interface Ticket #5894
07-15-19	1.2.0	UPDATE - Ensure that the minimum number of manager nodes have been created in a swarm Ticket #5893
07-15-19	1.2.0	UPDATE - Ensure that management plane traffic is separated from data plane traffic Ticket #5897
07-15-19	1.2.0	UPDATE - Ensure auditing is configured for the Docker daemon Ticket #6157
07-15-19	1.2.0	REMOVE - --disable-legacy-registry is deprecated Ticket #6697
07-15-19	1.2.0	UPDATE - Ensure that all Docker swarm overlay networks are encrypted Ticket #5895

07-15-19	1.2.0	UPDATE - Ensure that Docker's secret management commands are used for managing secrets in a swarm cluster Ticket #5896
07-15-19	1.2.0	UPDATE - Ensure that CA certificates are rotated as appropriate Ticket #5898
07-15-19	1.2.0	UPDATE - Ensure swarm mode is not Enabled, if not needed Ticket #5892
07-15-19	1.2.0	ADD - additional profiles Ticket #6026
07-15-19	1.2.0	UPDATE - Ensure that image sprawl is avoided Ticket #5899
07-15-19	1.2.0	UPDATE - Ensure that a user for the container has been created Ticket #8178
07-15-19	1.2.0	UPDATE - Ensure auditing is configured for Docker files and directories - /etc/default/docker Ticket #8003
07-15-19	1.2.0	UPDATE - Ensure swarm mode is not Enabled, if not needed Ticket #5891
07-15-19	1.2.0	UPDATE - Ensure auditing is configured for Docker files and directories - /var/lib/docker Ticket #8073
07-15-19	1.2.0	UPDATE - Ensure a separate partition for containers has been created Ticket #8227
07-15-19	1.2.0	UPDATE - Ensure secrets are not stored in Dockerfiles Ticket #8517

07-15-19	1.2.0	UPDATE - Ensure that swarm manager is run in auto-lock mode Ticket #8544
07-15-19	1.2.0	UPDATE - Ensure sshd is not run within containers Ticket #8490
07-15-19	1.2.0	UPDATE - Ensure mount propagation mode is not set to shared Ticket #8521
07-15-19	1.2.0	UPDATE - Ensure images are scanned and rebuilt to include security patches Ticket #8513
07-15-19	1.2.0	UPDATE - Ensure that the /etc/default/docker file ownership is set to root:root Ticket #8491
07-15-19	1.2.0	ADD - additional profiles Ticket #6026
07-15-19	1.2.0	UPDATE - Ensure swarm mode is not Enabled, if not needed Ticket #5891