Center for Internet Security®

CIS Docker 1.13.0 Benchmark

v1.0.0 - 01-19-2017

Table of Contents

Overview

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 1.13.0 or later technology.

Intended Audience

This document, CIS Docker 1.13.0 Benchmark, provides prescriptive guidance for establishing a secure configuration posture for Docker container version 1.13.0. This guide was tested against Docker 1.13.0 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.

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://community.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**

  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 2 - Docker**

  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:

Author
Pravin Goyal

Contributor
Thomas Sjögren
Rory McCune, NCC Group PLC
Diogo Monica, Docker, Inc.
Manideep Konakandla, Adobe
Jordan Rakoske

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 Create a separate partition for containers (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` would be mounted under `/` or `/var` partitions based on availability.

Rationale:

Docker depends on `/var/lib/docker` as the default directory where all Docker related files, including the images, are stored. This directory might fill up fast and soon Docker and the host could become unusable. So, it is advisable to create a separate partition (logical volume) for storing Docker files.

Audit:

At the Docker host execute the below command:

```
grep /var/lib/docker /etc/fstab
```

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

Remediation:

For new installations, create a separate partition for `/var/lib/docker` mount point. For systems that were previously installed, use the Logical Volume Manager (LVM) to create partitions.

Impact:

None.

Default Value:

By default, `/var/lib/docker` would be mounted under `/` or `/var` partitions based on availability.

References:

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

1.2 Harden the container host (Not Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Containers run on a Linux host. A container host can 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 would ensure that the 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 does current host system comply with. Ensure that the host systems actually comply with that host specific security benchmark.

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.

Additionally, you can run a kernel with `grsecurity` and `PaX`. This would add many safety checks, both at compile-time and run-time. It is also designed to defeat many exploits and has powerful security features. These features do not require Docker-specific configuration, since those security features apply system-wide, independent of containers.

Impact:

None.

Default Value:

By default, host has factory settings. It is not hardened.

References:

1. https://docs.docker.com/articles/security/
2. https://benchmarks.cisecurity.org/downloads/multiform/index.cfm
3. http://docs.docker.com/articles/security/#other-kernel-security-features
4. https://grsecurity.net/
5. https://en.wikibooks.org/wiki/Grsecurity
6. https://pax.grsecurity.net/
7. http://en.wikipedia.org/wiki/PaX

1.3 Keep Docker up to date (Not Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

There are frequent releases for Docker software that address security vulnerabilities, product bugs and bring in new functionality. Keep a tab on these product updates and upgrade as frequently as when new security vulnerabilities are fixed or deemed correct for your organization.

Rationale:

By staying up to date on Docker updates, vulnerabilities in the Docker software can be mitigated. An educated attacker may exploit known vulnerabilities when attempting to attain access or elevate privileges. Not installing regular Docker updates may leave you with running vulnerable Docker software. It might lead to elevation privileges, unauthorized access or other security breaches. Keep a track of new releases and update as necessary.

Audit:

Execute the below command and verify that the Docker version is up to date as deemed necessary. It is not a mandate to be on the latest one, though.

```
docker version
```

Remediation:

Keep a track of Docker releases and update as necessary.

Impact:

None.

Default Value:

Not Applicable

References:

1. https://docs.docker.com/installation/
2. https://github.com/docker/docker/releases/latest

1.4 Only allow trusted users to control Docker daemon (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

The Docker daemon currently requires '`root`' privileges. A user added to the '`docker`' group gives him full '`root`' access rights.

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 will then be able to alter your host file system without any restrictions. In simple terms, it means that you can attain elevated privileges with just being a member of the '`docker`' group and then starting a container with mapped `/` directory on the host.

Audit:

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

```
getent group docker
```

Remediation:

Remove any users from the '`docker`' group that are not trusted. Additionally, do not create a mapping of sensitive directories on host to container volumes.

Impact:

Rights to build and execute containers as normal user would be restricted.

Default Value:

Not Applicable

References:

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

1.5 Audit docker daemon (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit all Docker daemon activities.

Rationale:

Apart from auditing your regular Linux file system and system calls, audit Docker daemon as well. Docker daemon runs with '`root`' privileges. It is thus necessary to audit its activities and usage.

Audit:

Verify that there is an audit rule for Docker daemon. For example, execute below command:

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

This should list a rule for Docker daemon.

Remediation:

Add a rule for Docker daemon.

For example,

Add the line as below line in `/etc/audit/audit.rules` file:

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

Then, restart the audit daemon. For example,

```
service auditd restart
```

Impact:

Auditing generates quite big log files. Ensure to rotate and archive them periodically. Also, create a separate partition of audit to avoid filling root file system.

Default Value:

By default, 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

1.6 Audit Docker files and directories - /var/lib/docker (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `/var/lib/docker`.

Rationale:

Apart from auditing your regular Linux file system and system calls, audit all Docker related files and directories. Docker daemon runs with '`root`' privileges. Its behavior depends on some key files and directories. `/var/lib/docker` is one such directory. It holds all the information about containers. It must be audited.

Audit:

Verify that there is an audit rule corresponding to `/var/lib/docker` directory.

For example, execute below command:

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

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

Remediation:

Add a rule for `/var/lib/docker` directory.

For example,

Add the line as below in `/etc/audit/audit.rules` file:

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

Then, restart the audit daemon. For example,

```
service auditd restart
```

Impact:

Auditing generates quite big log files. Ensure to rotate and archive them periodically. Also, create a separate partition of audit to avoid filling root file system.

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

1.7 Audit Docker files and directories - /etc/docker (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `/etc/docker`.

Rationale:

Apart from auditing your regular Linux file system and system calls, audit all Docker related files and directories. Docker daemon runs with '`root`' privileges. Its behavior depends on some key files and directories. `/etc/docker` is one such directory. It holds various certificates and keys used for TLS communication between Docker daemon and Docker client. It must be audited.

Audit:

Verify that there is an audit rule corresponding to `/etc/docker` directory.

For example, execute below command:

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

This should list a rule for `/etc/docker` directory.

Remediation:

Add a rule for `/etc/docker` directory.

For example,

Add the line as below in `/etc/audit/audit.rules` file:

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

Then, restart the audit daemon. For example,

```
service auditd restart
```

Impact:

Auditing generates quite big log files. Ensure to rotate and archive them periodically. Also, create a separate partition of audit to avoid filling root file system.

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

1.8 Audit Docker files and directories - docker.service (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `docker.service`, if applicable.

Rationale:

Apart from auditing your regular Linux file system and system calls, audit all Docker related files and directories. Docker daemon runs with '`root`' privileges. Its behavior depends on some key files and directories. `docker.service` is one such file. The `docker.service` file might be present if the daemon parameters have been changed by an administrator. It holds various parameters for Docker daemon. It must be audited, if applicable.

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, verify that there is an audit rule corresponding to the file:

For example, execute the below command:

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

This should list a rule for `docker.service` as per its location.

Remediation:

If the file exists, add a rule for it.

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 generates quite big log files. Ensure to rotate and archive them periodically. Also, create a separate partition of audit to avoid filling root file system.

Default Value:

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

References:

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

1.9 Audit Docker files and directories - docker.socket (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `docker.socket`, if applicable.

Rationale:

Apart from auditing your regular Linux file system and system calls, audit all Docker related files and directories. Docker daemon runs with '`root`' privileges. Its behavior depends on some key files and directories. `docker.socket` is one such file. It holds various parameters for Docker daemon socket. It must be audited, if applicable.

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, verify that there is an audit rule corresponding to the file:

For example, execute the below command:

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

This should list a rule for `docker.socket` as per its location.

Remediation:

If the file exists, add a rule for it.

For example,

Add the line as below in `/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 generates quite big log files. Ensure to rotate and archive them periodically. Also, create a separate partition of audit to avoid filling root file system.

Default Value:

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

References:

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

1.10 Audit Docker files and directories - /etc/default/docker (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

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

Rationale:

Apart from auditing your regular Linux file system and system calls, audit all Docker related files and directories. Docker daemon runs with '`root`' privileges. Its behavior depends on some key files and directories. `/etc/default/docker` is one such file. It holds various parameters for Docker daemon. It must be audited, if applicable.

Audit:

Verify that there is an audit rule corresponding to `/etc/default/docker` file.

For example, execute below command:

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

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

Remediation:

Add a rule for `/etc/default/docker` file.

For example,

Add the line as below in `/etc/audit/audit.rules` file:

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

Then, restart the audit daemon. For example,

```
service auditd restart
```

Impact:

Auditing generates quite big log files. Ensure to rotate and archive them periodically. Also, create a separate partition of audit to avoid filling root file system.

Default Value:

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

1.11 Audit 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:

Apart from auditing your regular Linux file system and system calls, audit all Docker related files and directories. Docker daemon runs with '`root`' privileges. Its behavior depends on some key files and directories. `/etc/docker/daemon.json` is one such file. It holds various parameters for Docker daemon. It must be audited, if applicable.

Audit:

Verify that there is an audit rule corresponding to `/etc/docker/daemon.json` file.

For example, execute below command:

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

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

Remediation:

Add a rule for `/etc/docker/daemon.json` file.

For example,

Add the line as below in `/etc/audit/audit.rules` file:

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

Then, restart the audit daemon. For example,

```
service auditd restart
```

Impact:

Auditing generates quite big log files. Ensure to rotate and archive them periodically. Also, create a separate partition of audit to avoid filling root file system.

Default Value:

By default, Docker related files and directories are not audited. The file `/etc/docker/daemon.json` may not be available on the system. 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/daemon/#daemon-configuration-file

1.12 Audit Docker files and directories - /usr/bin/docker-containerd (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `/usr/bin/docker-containerd`, if applicable.

Rationale:

Apart from auditing your regular Linux file system and system calls, audit all Docker related files and directories. Docker daemon runs with '`root`' privileges. Its behavior depends on some key files and directories. `/usr/bin/docker-containerd` is one such file. Docker now relies on `containerd` and `runC` to spawn containers. It must be audited, if applicable.

Audit:

Verify that there is an audit rule corresponding to `/usr/bin/docker-containerd` file.

For example, execute below command:

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

This should list a rule for `/usr/bin/docker-containerd` file.

Remediation:

Add a rule for `/usr/bin/docker-containerd` file.

For example,

Add the line as below in `/etc/audit/audit.rules` file:

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

Then, restart the audit daemon. For example,

```
service auditd restart
```

Impact:

Auditing generates quite big log files. Ensure to rotate and archive them periodically. Also, create a separate partition of audit to avoid filling root file system.

Default Value:

By default, Docker related files and directories are not audited. The file `/usr/bin/docker-containerd` may not be available on the system. 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/

1.13 Audit Docker files and directories - /usr/bin/docker-runc (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `/usr/bin/docker-runc`, if applicable.

Rationale:

Apart from auditing your regular Linux file system and system calls, audit all Docker related files and directories. Docker daemon runs with '`root`' privileges. Its behavior depends on some key files and directories. `/usr/bin/docker-runc` is one such file. Docker now relies on `containerd` and `runC` to spawn containers. It must be audited, if applicable.

Audit:

Verify that there is an audit rule corresponding to `/usr/bin/docker-runc` file.

For example, execute below command:

```
auditctl -l | grep /usr/bin/docker-runc
```

This should list a rule for `/usr/bin/docker-runc` file.

Remediation:

Add a rule for `/usr/bin/docker-runc` file.

For example,

Add the line as below in `/etc/audit/audit.rules` file:

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

Then, restart the audit daemon. For example,

```
service auditd restart
```

Impact:

Auditing generates quite big log files. Ensure to rotate and archive them periodically. Also, create a separate partition of audit to avoid filling root file system.

Default Value:

By default, Docker related files and directories are not audited. The file `/usr/bin/docker-runc` may not be available on the system. 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

2 Docker daemon configuration

This section lists the recommendations that alter and secure the behavior of Docker daemon (server). 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` or `/etc/default/docker` on Debian and Ubuntu. 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 Restrict network traffic between containers (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

By default, all network traffic is allowed between containers on the same host. If not desired, restrict all the inter container communication. Link specific containers together that require inter communication.

Rationale:

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

Audit:

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

Ensure that the '`--icc`' parameter is set to '`false`' or verify by running the below command.

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

It should return com.docker.network.bridge.enable_icc:false.

Remediation:

Run the docker in daemon mode and pass '`--icc=false`' as argument.

For Example,

```
/usr/bin/dockerd --icc=false
```

Impact:

The inter container communication would be disabled. No containers would be able to talk to another container on the same host. If any communication between containers on the same host is desired, then it needs to be explicitly defined using container linking.

Default Value:

By default, all inter container communication is allowed.

References:

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

2.2 Set the logging level (Scored)

Profile Applicability:

- Level 1 - Docker

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:

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

Ensure that either the '`--log-level`' parameter is not present or if present, then it is set to '`info`'.

Remediation:

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/engine/reference/commandline/daemon/

2.3 Allow Docker to make changes to iptables (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Iptables are used to set up, maintain, and inspect the tables of IP packet filter rules in the Linux kernel. Allow the Docker daemon to make changes to the `iptables`.

Rationale:

Docker will never make changes to your system `iptables` rules if you choose to do so. Docker server would automatically make the needed changes to iptables based on how you choose your networking options for the containers if it is allowed to do so. It is recommended to let Docker server make changes to `iptables` automatically to avoid networking misconfiguration that might hamper the communication between containers and to the outside world. Additionally, it would save you hassles of updating `iptables` every time you choose to run the containers or modify networking options.

Audit:

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

Ensure that the '`--iptables`' parameter is either not present or not set to '`false`'.

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:

None.

Default Value:

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

References:

1. https://docs.docker.com/v1.8/articles/networking/

2.4 Do not use insecure registries (Scored)

Profile Applicability:

- Level 1 - Docker

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 the one not having either valid registry certificate or is not using TLS. You should not be using any insecure registries in the production environment. Insecure registries can be tampered with leading to possible compromise to your production system.

Additionally, If a registry is marked as insecure then '`docker pull`', '`docker push`', and '`docker search`' commands will not result in an error message and the user might be indefinitely working with insecure registries without ever being notified of potential danger.

Audit:

Run `docker info` or execute the below command to find out if any insecure registries are used:

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

Ensure that the '`--insecure-registry`' parameter is not present.

Remediation:

Do not use any insecure registries.

For example, do not start the Docker daemon as below:

```
dockerd --insecure-registry 10.1.0.0/16
```

Impact:

None.

Default Value:

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

References:

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

2.5 Do not use the aufs storage driver (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

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

Rationale:

The '`aufs`' storage driver is the oldest storage driver. It is based on a Linux kernel patch-set that is unlikely to be merged into the main Linux kernel. '`aufs`' driver is also known to cause some serious kernel crashes. '`aufs`' just has legacy support from 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 | grep -e "^Storage Driver:\s*aufs\s*$"
```

The above command should not return anything.

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.

Default Value:

By default, Docker uses '`devicemapper`' as the storage driver on most of the platforms. Default storage driver can vary based on your OS vendor. You should use the storage driver that is best supported by your preferred vendor.

References:

1. http://docs.docker.com/reference/commandline/cli/#daemon-storage-driver-option
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/

2.6 Configure TLS authentication for Docker daemon (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

It is possible to make the Docker daemon to listen on a specific IP and port and any other Unix socket other than default Unix socket. Configure TLS authentication to restrict access to Docker daemon via IP and port.

Rationale:

By default, 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 can have full access to Docker daemon and in turn to the host system. Hence, 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, configure TLS authentication for the daemon and Docker Swarm APIs (if using). This would restrict the connections to your Docker daemon over the network to a limited number of clients who could successfully authenticate over TLS.

Audit:

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

Ensure that the below parameters are present:

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

Remediation:

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

Impact:

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

Default Value:

By default, TLS authentication is not configured.

References:

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

2.7 Set default ulimit as appropriate (Not Scored)

Profile Applicability:

- Level 1 - Docker

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 started by it. Setting system resource limits judiciously saves you from many disasters such as a fork bomb. Sometimes, even friendly users and legitimate processes can overuse system resources and in-turn can make the system unusable.

Setting default ulimit for the Docker daemon would enforce the ulimit for all container instances. You would not need to setup ulimit for each container instance. However, the default ulimit can be overridden during container runtime, if needed. Hence, to control the system resources, define a default ulimit as needed in your environment.

Audit:

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

Ensure that the '`--default-ulimit`' parameter is set as appropriate.

Remediation:

Run the docker in daemon mode and pass '`--default-ulimit`' as argument with respective ulimits as appropriate in your environment.

For Example,

```
dockerd --default-ulimit nproc=1024:2408 --default-ulimit nofile=100:200
```

Impact:

If the ulimits are not set properly, the desired resource control might not be achieved and might even make the system unusable.

Default Value:

By default, no ulimit is set.

References:

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

2.8 Enable user namespace support (Scored)

Profile Applicability:

- Level 2 - Docker

Description:

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

Rationale:

The Linux kernel user namespace support in 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 will have expected administrative privilege 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 would find the PID of the container and then would list the host user associated with the container process. If the container process is running as `root`, then this recommendation is non-compliant.

Alternatively, you can run `docker info` to ensure that the `userns` is listed under `Security Options:`

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

Remediation:

Please consult 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 does not work automatically. You might have to create your own mapping.

However, 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 makes quite a few Docker features incompatible and also currently breaks a few functionalities. Check out the Docker documentation and referenced links for details.

Default Value:

By default, user namespace is not remapped.

References:

1. http://man7.org/linux/man-pages/man7/user_namespaces.7.html
2. https://docs.docker.com/engine/reference/commandline/daemon/
3. http://events.linuxfoundation.org/sites/events/files/slides/User%20Namespaces%20-%20ContainerCon%202015%20-%2016-9-final_0.pdf
4. https://github.com/docker/docker/issues/21050

2.9 Confirm default cgroup usage (Scored)

Profile Applicability:

- Level 2 - Docker

Description:

The `--cgroup-parent` option allows you to set the default cgroup parent to use for all the containers. If there is no specific use case, this 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 that is the default. This usage should be monitored and confirmed. By attaching to a different cgroup than the one that is a default, it is possible to share resources unevenly and thus might starve the host for resources.

Audit:

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

Ensure that the '`--cgroup-parent`' parameter is either not set or is set as appropriate non-default cgroup.

Remediation:

The default setting is good enough and can be left as-is. If you want to specifically set a non-default cgroup, pass `--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/daemon/

2.10 Do not change base device size until needed (Scored)

Profile Applicability:

- Level 2 - Docker

Description:

In certain circumstances, you might need containers bigger than 10G in size. In these cases, carefully choose the base device size.

Rationale:

The base device size can be increased at 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 inherited by pulled images.

Though the file system does not allot the increased size if it is empty, it will use more space for the empty case depending upon the device size. This may cause a denial of service by ending up in file system being over-allocated or full.

Audit:

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

Execute the above command and it should not show any `--storage-opt dm.basesize` parameters.

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/daemon/#storage-driver-options

2.11 Use authorization plugin (Scored)

Profile Applicability:

- Level 2 - Docker

Description:

Use authorization plugin to manage access to Docker daemon.

Rationale:

Docker's out-of-the-box authorization model is all or nothing. Any user with permission to access the Docker daemon can run any Docker client command. The same is true for callers using Docker's remote 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 Docker daemon.

Audit:

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

Ensure that the '`--authorization-plugin`' parameter is set as appropriate.

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 specifically passes through authorization plugin mechanism. This might introduce a slight performance drop.

Default Value:

By default, authorization plugins are not set up.

References:

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

2.12 Configure centralized and remote logging (Scored)

Profile Applicability:

- Level 2 - Docker

Description:

Docker now supports various log drivers. A preferable way to store logs is the one that supports centralized and remote logging.

Rationale:

Centralized and remote logging ensures that all important log records are safe despite catastrophic events. Docker now supports various such logging drivers. Use the one that suits your environment the best.

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. Ensure that it is set as appropriate.

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

Remediation:

Step 1: Setup the desired log driver by following its documentation.

Step 2: Start the docker daemon with 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/

2.13 Disable operations on legacy registry (v1) (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

The latest Docker registry is v2. All operations on the legacy registry version (v1) should be restricted.

Rationale:

Docker registry v2 brings in many performance and security improvements over v1. It supports container image provenance and other security features such as image signing and verification. Hence, operations on Docker legacy registry should be restricted.

Audit:

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

The above command should list `--disable-legacy-registry` as an option passed to the docker daemon.

Remediation:

Start the docker daemon as below:

```
dockerd --disable-legacy-registry
```

Impact:

Legacy registry operations would be restricted.

Default Value:

By default, legacy registry operations are allowed.

References:

1. https://docs.docker.com/engine/reference/commandline/daemon/
2. https://github.com/docker/docker/issues/8093
3. https://github.com/docker/docker/issues/9015
4. https://github.com/docker/docker-registry/issues/612
5. https://docs.docker.com/registry/spec/api/
6. https://the.binbashtheory.com/creating-private-docker-registry-2-0-with-token-authentication-service/
7. https://blog.docker.com/2015/07/new-tool-v1-registry-docker-trusted-registry-v2-open-source/
8. http://www.slideshare.net/Docker/docker-registry-v2

2.14 Enable live restore (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

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

Rationale:

One of the important security triads is availability. Setting '`--live-restore`' flag in the docker daemon ensures that container execution is not interrupted when the docker daemon is not available. This also means that it is now easier to update and patch the docker daemon without execution downtime.

Audit:

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

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

Alternatively run the below command and ensure that --live-restore is used.

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

Remediation:

Run the docker in daemon mode and pass '`--live-restore`' as an argument.

For Example,

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

Impact:

None.

Default Value:

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

References:

1. https://github.com/docker/docker/pull/23213

2.15 Do not enable swarm mode, if not needed (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Do not enable swarm mode on a docker engine instance unless 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 increase its attack surface and this should be avoided unless required.

Audit:

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. Confirm if swarm mode on the docker engine instance is actually needed.

Remediation:

If swarm mode has been enabled on a system in error, run

docker swarm leave

Impact:

None.

Default Value:

By default, docker swarm mode is not enabled.

References:

1. https://docs.docker.com/engine/reference/commandline/swarm_init/

2.16 Control the number of manager nodes in a swarm (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

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 change its configuration 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 practical odd number to achieve the appropriate level of tolerance should be configured.

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 can be demoted as worker using the following command:

```
docker node demote <ID>
```

Where <ID> is the node ID value of the manager to be demoted.

Impact:

None

Default Value:

A single manager is all that 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

2.17 Bind swarm services to a specific host interface (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

By default, the docker swarm services will listen to all interfaces on the host, which 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 the swarm services will listen on all interfaces on the host. If a host has multiple network interfaces this may be undesirable as it may expose the docker swarm services to networks which are not involved in 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:

List the network listener on port 2377/TCP (the default for docker swarm) and confirm that it is only listening on specific interfaces. For example, using ubuntu this could be done with the following command:

```
netstat -lt | grep -i 2377
```

Remediation:

Remediation of this 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-value
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.

2.18 Disable Userland Proxy (Scored)

Profile Applicability:

- Level 1 - Docker

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:

Docker engine provides two mechanisms for forwarding ports from the host to containers, hairpin NAT, and 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 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:

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

Ensure that the `--userland-proxy` parameter is set to `false`.

Remediation:

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/

2.19 Encrypt data exchanged between containers on different nodes on the overlay network (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Encrypt data exchanged between containers on different nodes on the overlay network.

Rationale:

By default, data exchanged between containers on different nodes on the overlay network is not encrypted. This could potentially expose traffic between the container nodes.

Audit:

Run the below command and ensure that each overlay network has been `encrypted`.

```
docker network ls --filter driver=overlay --quiet | xargs docker network inspect --
format '{{.Name}} {{ .Options }}'
```

Remediation:

Create overlay network with `--opt encrypted` flag.

Impact:

None

Default Value:

By default, data exchanged between containers on different nodes on the overlay network are not encrypted in the Docker swarm mode.

References:

1. https://docs.docker.com/engine/userguide/networking/overlay-security-model/
2. https://github.com/docker/docker/issues/24253

2.20 Apply a daemon-wide custom seccomp profile, if needed (Not Scored)

Profile Applicability:

- Level 2 - Docker

Description:

You can choose to apply your custom seccomp profile at the daemon-wide level if needed and override Docker's default seccomp profile.

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 the applications do not need all the system calls and thus benefit by having a reduced set of available system calls. The reduced set of system calls reduces the total kernel surface exposed to the application and thus improvises application security.

You could apply your own custom seccomp profile instead of Docker's default seccomp profile. Alternatively, if Docker's default profile is good for your environment, you can choose to ignore this recommendation.

Audit:

Run the below command and review the seccomp profile listed in the `Security Options` section. If it is `default`, that means, Docker's default seccomp profile is applied.

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

Remediation:

By default, Docker's default seccomp profile is applied. If this is good 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. These address some critical

vulnerabilities/issues within container environments (for example, kernel key ring calls). So, you should be very careful while overriding the defaults.

Default Value:

By default, Docker applies a seccomp profile.

References:

1. https://github.com/docker/docker/pull/26276

2.21 Avoid experimental features in production (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Avoid experimental features in production.

Rationale:

Experimental is now a runtime docker daemon flag instead of a separate build. Passing `--experimental` as a runtime flag to the docker daemon, activates experimental features. Experimental is now considered a stable release, but with a couple of features which might not have tested and guaranteed API stability.

Audit:

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

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

Remediation:

Do not pass `--experimental` as a runtime parameter to the docker daemon.

Impact:

None

Default Value:

By default, experimental features are not activated on the docker daemon.

References:

1. https://github.com/docker/docker/issues/26713
2. https://github.com/docker/docker/pull/27223

2.22 Use Docker's secret management commands for managing secrets in a Swarm cluster (Not Scored)

Profile Applicability:

- Level 2 - Docker

Description:

Use Docker's in-built secret management command.

Rationale:

Docker has various commands for managing secrets in a Swarm cluster. This is the foundation for future secret support in Docker with potential improvements such as Windows support, different backing stores, etc.

Audit:

On a swarm manager node, run the below command and ensure `docker secret` management is used in your environment, if applicable.

```
docker secret ls
```

Remediation:

Follow `docker secret` documentation and use it to manage secrets effectively.

Impact:

None

Default Value:

Not Applicable

References:

1. https://github.com/docker/docker/pull/27794

2.23 Run swarm manager in auto-lock mode (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

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 should 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 swarm with `--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.

Audit:

Run the below command. If it outputs the key, it means 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 and is non-compliant with respect to this recommendation.

```
docker swarm unlock-key
```

Remediation:

If you are initializing swarm, use the below command.

```
docker swarm init --autolock
```

If you want to set `--autolock` on an existing swarm manager node, use the below command.

```
docker swarm update --autolock
```

Impact:

A swarm in auto-lock mode won't recover from a re-start without manual intervention from a user to enter the unlock key. In some deployments, this might not be good for availability.

Default Value:

By default, swarm manager does not run in auto-lock mode.

References:

1. https://github.com/mstanleyjones/docker.github.io/blob/af7dfdba8504f9b102fb31a78cd08a06c33a8975/engine/swarm/swarm_manager_locking.md

2.24 Rotate swarm manager auto-lock key periodically (Not Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Rotate swarm manager auto-lock key periodically.

Rationale:

Swarm manager auto-lock key is not automatically rotated. You should rotate them periodically as a best practice.

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 if there is a key rotation record and the keys were rotated at a pre-defined frequency.

Remediation:

Run the below command to rotate the keys.

```
docker swarm unlock-key --rotate
```

Additionally, to facilitate audit for this recommendation, 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://github.com/mstanleyjones/docker.github.io/blob/af7dfdba8504f9b102fb31a78cd08a06c33a8975/engine/swarm/swarm_manager_locking.md

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 Verify that docker.service file ownership is set to root:root (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Verify that the '`docker.service`' file ownership and group-ownership are correctly set to '`root`'.

Rationale:

'`docker.service`' file contains sensitive parameters that may alter the behavior of Docker daemon. Hence, it should be owned and group-owned by '`root`' to maintain the integrity of the file.

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 below command with 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.service | grep -v root:root
```

The above command 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 exists, execute the below command with 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.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 ownership and group-ownership for this file is correctly set to '`root`'.

References:

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

3.2 Verify that docker.service file permissions are set to 644 or more restrictive (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Verify that the '`docker.service`' file permissions are correctly set to '`644`' or more restrictive.

Rationale:

'`docker.service`' file contains sensitive parameters that may alter the behavior of Docker daemon. Hence, it should not be writable by any other user other than '`root`' to maintain the integrity of the file.

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 below command with the correct file path to verify that the file permissions are set to '`644`' or more restrictive.

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 below command with 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/

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

Profile Applicability:

- Level 1 - Docker

Description:

Verify that the '`docker.socket`' file ownership and group ownership is correctly set to '`root`'.

Rationale:

'`docker.socket`' file contains sensitive parameters that may alter the behavior of Docker remote API. Hence, it should be owned and group-owned by '`root`' to maintain the integrity of the file.

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 below command with 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 above command should not return anything.

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 below command with 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 this file is correctly set to '`root`'.

References:

1. https://docs.docker.com/articles/basics/#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/

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

Profile Applicability:

- Level 1 - Docker

Description:

Verify that the '`docker.socket`' file permissions are correctly set to '`644`' or more restrictive.

Rationale:

'`docker.socket`' file contains sensitive parameters that may alter the behavior of Docker remote API. Hence, it should be writable only by '`root`' to maintain the integrity of the file.

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 below command with the correct file path to verify that the file permissions are set to '`644`' or more restrictive.

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 exists, execute the below command with 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. In that case, this recommendation is not applicable. By default, if the file is present, the file permissions for this file are correctly set to '`644`'.

References:

1. https://docs.docker.com/articles/basics/#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/

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

Profile Applicability:

- Level 1 - Docker

Description:

Verify that the `/etc/docker` directory ownership and group-ownership is correctly set to '`root`'.

Rationale:

'`/etc/docker`' directory contains certificates and keys in addition to various sensitive files. Hence, it should be owned and group-owned by '`root`' to maintain the integrity of the directory.

Audit:

Execute the below command to verify that the directory is owned and group-owned by '`root`':

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

The above command should not return anything.

Remediation:

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

This would set 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/articles/certificates/

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

Profile Applicability:

- Level 1 - Docker

Description:

Verify that the `/etc/docker` directory permissions are correctly set to '`755`' or more restrictive.

Rationale:

'`/etc/docker`' directory contains certificates and keys in addition to various sensitive files. Hence, it should only be writable by '`root`' to maintain the integrity of the directory.

Audit:

Execute the below command to verify that the directory has permissions of '`755`' or more restrictive:

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

Remediation:

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

This would set the permissions for the directory to '`755`'.

Impact:

None.

Default Value:

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

References:

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

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

Profile Applicability:

- Level 1 - Docker

Description:

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

Rationale:

`/etc/docker/certs.d/<registry-name>` directory contains Docker registry certificates. These certificate files must be owned and group-owned by '`root`' to maintain the integrity of the certificates.

Audit:

Execute the below command to verify that the registry certificate files are owned and group-owned by '`root`':

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

The above command should not return anything.

Remediation:

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

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

Impact:

None.

Default Value:

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

References:

1. https://docs.docker.com/articles/certificates/
2. http://docs.docker.com/reference/commandline/cli/#insecure-registries

3.8 Verify that registry certificate file permissions are set to 444 or more restrictive (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Verify that all the registry certificate files (usually found under `/etc/docker/certs.d/<registry-name>` directory) have permissions of '`444`' or more restrictive.

Rationale:

`/etc/docker/certs.d/<registry-name>` directory contains Docker registry certificates. These certificate files must have permissions of '`444`' to maintain the integrity of the certificates.

Audit:

Execute the below command to verify that the registry certificate files have permissions of '`444`' or more restrictive:

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

Remediation:

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

This would set the permissions for 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.

References:

1. https://docs.docker.com/articles/certificates/
2. http://docs.docker.com/reference/commandline/cli/#insecure-registries

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

Profile Applicability:

- Level 1 - Docker

Description:

Verify that the TLS CA certificate file (the file that is passed alongwith '`--tlscacert`' parameter) is owned and group-owned by '`root`'.

Rationale:

The TLS CA certificate file should be protected from any tampering. It is used to authenticate Docker server based on given CA certificate. Hence, it must be owned and group-owned by '`root`' to maintain the integrity of the CA certificate.

Audit:

Execute the below command 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 not return anything.

Remediation:

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

This would set the 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/articles/certificates/
2. http://docs.docker.com/articles/https/

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

Profile Applicability:

- Level 1 - Docker

Description:

Verify that the TLS CA certificate file (the file that is passed alongwith '`--tlscacert`' parameter) has permissions of '`444`' or more restrictive.

Rationale:

The TLS CA certificate file should be protected from any tampering. It is used to authenticate Docker server based on given CA certificate. Hence, it must have permissions of '`444`' to maintain the integrity of the CA certificate.

Audit:

Execute the below command to verify that the TLS CA certificate file has permissions of '`444`' or more restrictive:

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

Remediation:

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

This would set the file permissions of the TLS CA file to '`444`'.

Impact:

None.

Default Value:

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

References:

1. https://docs.docker.com/articles/certificates/
2. http://docs.docker.com/articles/https/

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

Profile Applicability:

- Level 1 - Docker

Description:

Verify that the Docker server certificate file (the file that is passed alongwith '`--tlscert`' parameter) is owned and group-owned by '`root`'.

Rationale:

The Docker server certificate file should be protected from any tampering. It is used to authenticate Docker server based on the given server certificate. Hence, it must be owned and group-owned by '`root`' to maintain the integrity of the certificate.

Audit:

Execute the below command to verify that the Docker server certificate file is owned and group-owned by '`root`':

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

The above command should not return anything.

Remediation:

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

This would set the ownership and 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/articles/certificates/
2. http://docs.docker.com/articles/https/

3.12 Verify that Docker server certificate file permissions are set to 444 or more restrictive (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Verify that the Docker server certificate file (the file that is passed alongwith '`--tlscert`' parameter) has permissions of '`444`' or more restrictive.

Rationale:

The Docker server certificate file should be protected from any tampering. It is used to authenticate Docker server based on the given server certificate. Hence, it must have permissions of '`444`' to maintain the integrity of the certificate.

Audit:

Execute the below command to verify that the Docker server certificate file has permissions of '`444`' or more restrictive:

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

Remediation:

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

This would set the file permissions of the Docker server file to '`444`'.

Impact:

None.

Default Value:

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

References:

1. https://docs.docker.com/articles/certificates/
2. http://docs.docker.com/articles/https/

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

Profile Applicability:

- Level 1 - Docker

Description:

Verify that the Docker server certificate key file (the file that is passed alongwith '`--tlskey`' parameter) is owned and group-owned by '`root`'.

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. Hence, it must be owned and group-owned by '`root`' to maintain the integrity of the Docker server certificate.

Audit:

Execute the below command to verify that the Docker server certificate key file is owned and group-owned by '`root`':

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

The above command should not return anything.

Remediation:

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

This would set the ownership and group-ownership for the Docker server certificate key file to '`root`'.

Impact:

None.

Default Value:

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

References:

1. https://docs.docker.com/articles/certificates/
2. http://docs.docker.com/articles/https/

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

Profile Applicability:

- Level 1 - Docker

Description:

Verify that the Docker server certificate key file (the file that is passed alongwith '`--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. Hence, it must have permissions of '`400`' to maintain the integrity of the Docker server certificate.

Audit:

Execute the below command to verify that the Docker server certificate key file has permissions of '`400`':

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

Remediation:

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

This would set the Docker server certificate key file permissions to '`400`'.

Impact:

None.

Default Value:

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

References:

1. https://docs.docker.com/articles/certificates/
2. http://docs.docker.com/articles/https/

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

Profile Applicability:

- Level 1 - Docker

Description:

Verify that the Docker socket file is owned by '`root`' and group-owned by '`docker`'.

Rationale:

Docker daemon runs as '`root`'. The default Unix socket hence must be owned by '`root`'. If any other user or process owns this socket, then it might be possible for that non-privileged user or process to interact with Docker daemon. Also, such a non-privileged user or process might interact with containers. This is neither secure nor desired behavior.

Additionally, the Docker installer creates a Unix group called '`docker`'. You can add users to this group, and then those users would be able to read and write to default Docker Unix socket. The membership to the '`docker`' group is tightly controlled by the system administrator. If any other group owns this socket, then it might be possible for members of that group to interact with Docker daemon. Also, such a group might not be as tightly controlled as the '`docker`' group. This is neither secure nor desired behavior.

Hence, the default Docker Unix socket file must be owned by '`root`' and group-owned by '`docker`' to maintain the integrity of the socket file.

Audit:

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 above command should not return anything.

Remediation:

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

This would set the ownership to '`root`' and group-ownership to '`docker`' for default Docker socket file.

Impact:

None.

Default Value:

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

References:

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

3.16 Verify that Docker socket file permissions are set to 660 or more restrictive (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Verify that the Docker socket file has permissions of '`660`' or more restrictive.

Rationale:

Only '`root`' and members of '`docker`' group should be allowed to read and write to default Docker Unix socket. Hence, the Docket socket file must have permissions of '`660`' or more restrictive.

Audit:

Execute the below command to verify that the Docker socket file has permissions of '`660`' or more restrictive:

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

Remediation:

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

This would set the file permissions of the Docker socket file to '`660`'.

Impact:

None.

Default Value:

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

References:

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

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

Profile Applicability:

- Level 1 - Docker

Description:

Verify that the '`daemon.json`' file ownership and group-ownership is correctly set to '`root`'.

Rationale:

`'daemon.json'` file contains sensitive parameters that may alter the behavior of docker daemon. Hence, it should be owned and group-owned by '`root`' to maintain the integrity of the file.

Audit:

Execute the below command 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 above command should not return anything.

Remediation:

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

This would set the ownership and group-ownership for the file to '`root`'.

Impact:

None.

Default Value:

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

References:

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

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

Profile Applicability:

- Level 1 - Docker

Description:

Verify that the '`daemon.json`' file permissions are correctly set to '`644`' or more restrictive.

Rationale:

'`daemon.json`' file contains sensitive parameters that may alter the behavior of docker daemon. Hence, it should be writable only by '`root`' to maintain the integrity of the file.

Audit:

Execute the below command to verify that the file permissions are correctly set to '`644`' or more restrictive:

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

Remediation:

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

This would set the file permissions for this file to '`644`'.

Impact:

None.

Default Value:

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

References:

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

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

Profile Applicability:

- Level 1 - Docker

Description:

Verify that the '`/etc/default/docker`' file ownership and group-ownership is correctly set to '`root`'.

Rationale:

'`/etc/default/docker`' file contains sensitive parameters that may alter the behavior of docker daemon. Hence, it should be owned and group-owned by '`root`' to maintain the integrity of the file.

Audit:

Execute the below command to verify that the file is owned and group-owned by '`root`':

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

The above command should not return anything.

Remediation:

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

This would set the ownership and group-ownership for the file to '`root`'.

Impact:

None.

Default Value:

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

References:

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

3.20 Verify that /etc/default/docker file permissions are set to 644 or more restrictive (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Verify that the '`/etc/default/docker`' file permissions are correctly set to '`644`' or more restrictive.

Rationale:

'`/etc/default/docker`' file contains sensitive parameters that may alter the behavior of docker daemon. Hence, it should be writable only by '`root`' to maintain the integrity of the file.

Audit:

Execute the below command to verify that the file permissions are correctly set to '`644`' or more restrictive:

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

Remediation:

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

This would set the file permissions for this file to '`644`'.

Impact:

None.

Default Value:

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

References:

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

4 Container Images and Build File

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 Create a user for the container (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Create a non-root user for the container in the Dockerfile for the container image.

Rationale:

It is a good practice to run the container as a non-root user, if possible. Though user namespace mapping is now available, if a user is already defined in the container image, the container is run as that user by default and specific user namespace remapping is not required.

Audit:

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

The above command should return container username or user ID. If it is blank it means, the container is running as `root`.

Remediation:

Ensure that the Dockerfile for the container image contains below instruction:

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

where username or ID refers to the user that could be found in the container base image. If there is no specific user created in the container base image, then add a `useradd` command to add the specific user before `USER` instruction.

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 the containers do not need, consider deleting them. After deleting those users, commit the image and then generate new instances of containers for use.

Impact:

None.

Default Value:

By default, the containers are run with `root` privileges and as user `root` 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
4. https://www.altiscale.com/hadoop-blog/making-docker-work-yarn/
5. http://docs.docker.com/articles/security/

4.2 Use trusted base images for containers (Not Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Ensure that the container image is written either from scratch or is based on another established and trusted base image downloaded over a secure channel.

Rationale:

Official repositories are Docker images curated and optimized by the Docker community or the vendor. There could be other potentially unsafe public repositories. You should thus exercise a lot of caution when obtaining container images.

Audit:

Inspect the Docker host by executing the below command:

```
docker images
```

This would list all the container images that are currently available for use on the Docker host. Interview the system administrator and obtain a proof of evidence that the list of images was obtained from trusted source over a secure channel.

Remediation:

Configure and use Docker Content trust.

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. http://docs.docker.com/reference/commandline/cli/#pull
6. https://github.com/docker/docker/pull/11109
7. https://blog.docker.com/2015/11/docker-trusted-registry-1-4/

4.3 Do not install unnecessary packages in the container (Not Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Containers tend to be minimal and slim down versions of the Operating System. Do not install anything that does not justify the purpose of container.

Rationale:

Bloating containers with unnecessary software could possibly increase the attack surface of the container. This also voids the concept of minimal and slim down versions of container images. Hence, do not install anything else apart from what is truly needed for the purpose of the container.

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 or equivalent command:

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

The above command would list the packages installed on the container. Review the list and ensure that it is legitimate.

Remediation:

At the outset, do not install anything on the container that does not justify the purpose. If the image had some packages that your container does not use, uninstall them.

Consider using a minimal base image rather than the standard Redhat/Centos/Debian images if you can. Some of the options include BusyBox and Alpine.

Not only does this trim your image size from >150Mb to ~20 Mb, there are also fewer tools and paths to escalate privileges. You can even remove the package installer as a final hardening measure for leaf/production containers.

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

4.4 Scan and rebuild the images to include security patches (Not Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Images should be scanned "frequently" for any vulnerabilities. Rebuild the images to include patches and then instantiate new containers from it.

Rationale:

Vulnerabilities are loopholes/bugs that can be exploited and security patches are updates to resolve these vulnerabilities. We can use image vulnerability scanning tools to find any kind of vulnerabilities within the images and then check for available patches to mitigate these vulnerabilities. Patches update the system to the most recent code base. Being on the current code base is important because that's where vendors focus on fixing problems. Evaluate the security patches before applying and follow the patching best practices.

Also, it would be better if, image vulnerability scanning tools could perform binary level analysis or hash based verification instead of just version string matching.

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 or equivalent command to find the list of packages installed within the container. Ensure that the security updates for various affected packages are installed.

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

Alternatively, you could run image vulnerability scanning tools which can scan all the images in your ecosystem and then apply patches for the detected vulnerabilities based on your patch management procedures.

Remediation:

Follow the below steps to rebuild the images with security patches:

Step 1: '`docker pull`' all the base images (i.e., given your set of Dockerfiles, extract all

images declared in '`FROM`' instructions, and re-pull them to check for an updated/patched versions). Patch the packages within the images too.
Step 2: Force a rebuild of each image with '`docker build --no-cache`'.
Step 3: Restart all containers with the updated images.

You could also use `ONBUILD` directive in the Dockerfile to trigger particular update instructions for images that you know are used as base images frequently.

Impact:

None

Default Value:

By default, containers and images are not updated of their own.

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

4.5 Enable Content trust for Docker (Scored)

Profile Applicability:

- Level 2 - Docker

Description:

Content trust is disabled by default. You should enable it.

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 integrity and publisher of specific image tags. This ensures provenance of container images.

Audit:

```
echo $DOCKER_CONTENT_TRUST
```

This should return 1.

Remediation:

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

```
export DOCKER_CONTENT_TRUST=1
```

Alternatively, 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 while working with images - `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 that defeats the purpose of enabling content trust and hence, 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

4.6 Add HEALTHCHECK instruction to the container image (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Add `HEALTHCHECK` instruction in your docker container images to perform the health check on running containers.

Rationale:

One of the important security triads is availability. Adding `HEALTHCHECK` instruction to your container image ensures that the docker engine periodically checks the running container instances against that instruction to ensure that the instances are still working.

Based on the reported health status, the docker engine could then exit non-working containers and instantiate new ones.

Audit:

Run the below command and ensure that the docker image has appropriate `HEALTHCHECK` instruction set up.

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

Remediation:

Follow Docker documentation and rebuild your container image with `HEALTHCHECK` instruction.

Impact:

None.

Default Value:

By default, `HEALTHCHECK` is not set.

References:

1. https://github.com/docker/docker/pull/22719

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

Profile Applicability:

- Level 1 - Docker

Description:

Do not use update instructions such as `apt-get update` alone or in a single line in the Dockerfile.

Rationale:

Adding the update instructions in a single line on the Dockerfile will cache the update layer. Thus, when you build any image later using the same instruction, previously cached update layer will be used. This could potentially deny any fresh updates to go in the later builds.

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 update instructions being in a single line:

```
docker history <Image_ID>
```

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

Remediation:

Use update instructions along with install instructions (or any other) and version pinning for packages while installing them. This would bust the cache and force to extract the required versions.

Alternatively, you could use `--no-cache` flag during `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/#/apt-get
2. https://github.com/docker/docker/issues/3313

4.8 Remove setuid and setgid permissions in the images (Not Scored)

Profile Applicability:

- Level 2 - Docker

Description:

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

Rationale:

setuid and setgid permissions could be used for elevating privileges. While these permissions are at times legitimately needed, these could potentially be used in privilege escalation attacks. Thus, you should consider dropping these permissions for the packages which do not need them within the images.

Audit:

Run the below command on the image to list the executables having setuid and setgid permissions:

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

Carefully, review the list and ensure that it is legitimate.

Remediation:

Allow setuid and setgid permissions only on executables which need them. You could remove these permissions during 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:

Above command breaks all the executables that depend on setuid or setgid permissions including the legitimate ones. Hence, be careful to modify the command to suit your requirements so that it does not drop the permissions of legitimate programs. This requires a careful examination of each executable and fine tuning the permissions.

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

4.9 Use COPY instead of ADD in Dockerfile (Not Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Use `COPY` instruction instead of `ADD` instruction in the Dockerfile.

Rationale:

`COPY` instruction just copies the files from the local host machine to the container file system. `ADD` instruction potentially could retrieve files from remote URLs and perform operations such as unpacking. Thus, `ADD` instruction introduces risks such as adding malicious files from URLs without scanning and unpacking procedure vulnerabilities.

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 `ADD` instructions:

```
docker history <Image_ID>
```

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

Remediation:

Use `COPY` instructions in Dockerfiles.

Impact:

You would need to take care of the functionalities provided by `ADD` instructions such as fetching files from remote URLs.

Default Value:

Not Applicable

References:

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

4.10 Do not store secrets in Dockerfiles (Not Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Do not store any secrets in Dockerfiles.

Rationale:

Dockerfiles could be backtracked easily by using native Docker commands such as `docker history` and various tools and utilities. Also, as a general practice, image publishers provide Dockerfiles to build the credibility for their images. Hence, the secrets within these Dockerfiles could be easily exposed and potentially be exploited.

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.

Impact:

You would need to identify a way to handle secrets for your Docker images.

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/

4.11 Install verified packages only (Not Scored)

Profile Applicability:

- Level 2 - Docker

Description:

Verify authenticity of the packages before installing them in the image.

Rationale:

Verifying authenticity of the packages is essential for building a secure container image. Tampered packages could potentially be malicious or have some known vulnerabilities that could be exploited.

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 how the authenticity of the packages is 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:

Use GPG keys for downloading and verifying packages or any other secure package distribution mechanism of your choice.

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

5 Container Runtime

The ways in which a container is started governs a lot of security implications. It is possible to provide potentially dangerous runtime parameters that might compromise the host and other containers on the host. Verifying container runtime is thus very important. Various recommendations to assess the container runtime are as below:

5.1 Do not disable AppArmor Profile (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

AppArmor is an effective and easy-to-use Linux application security system. It is available on quite a few Linux distributions by default such as Debian and Ubuntu.

Rationale:

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

Audit:

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

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

Remediation:

If AppArmor is applicable for your Linux OS, use it. You may have to follow below set of steps:

1. Verify if AppArmor is installed. If not, install it.
2. Create or import a AppArmor profile for Docker containers.
3. Put this profile in enforcing mode.
4. Start your Docker container using the customized AppArmor profile. For example,

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

Alternatively, you can keep the docker's default apparmor profile

Impact:

The container (process) would have set of restrictions as defined in AppArmor profile. If your AppArmor profile is mis-configured, then the container may not entirely work as expected.

Default Value:

By default, `docker-default` AppArmor profile is applied for running containers and this profile can be found at `/etc/apparmor.d/docker`.

References:

1. https://docs.docker.com/engine/security/apparmor/
2. http://docs.docker.com/articles/security/#other-kernel-security-features
3. https://github.com/docker/docker/blob/master/docs/security/apparmor.md
4. http://docs.docker.com/reference/run/#security-configuration

5.2 Verify SELinux security options, if applicable (Scored)

Profile Applicability:

- Level 2 - Docker

Description:

SELinux is an effective and easy-to-use Linux application security system. It is available on quite a few Linux distributions by default 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 thus add an extra layer of safety by enabling SELinux on your Linux host, if applicable.

Audit:

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

The above command should return all the security options currently configured for the containers.

Remediation:

If SELinux is applicable for your Linux OS, use it. You may have to follow below set of steps:

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:

The container (process) would have set of restrictions as defined in SELinux policy. If your SELinux policy is mis-configured, then the container may not entirely work as expected.

Default Value:

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

References:

1. http://docs.docker.com/articles/security/#other-kernel-security-features
2. http://docs.docker.com/reference/run/#security-configuration
3. http://docs.fedoraproject.org/en-US/Fedora/13/html/Security-Enhanced_Linux/

5.3 Restrict Linux Kernel Capabilities within containers (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

By default, Docker starts containers with a restricted set of Linux Kernel Capabilities. It means that any process may be granted the required capabilities instead of root access. Using Linux Kernel Capabilities, the processes do not have to run as root for almost all the specific areas where root privileges are usually needed.

Rationale:

Docker supports the addition and removal of capabilities, allowing use of a non-default profile. This may make Docker more secure through capability removal, or less secure through the addition of capabilities. It is thus recommended to remove all capabilities except those explicitly required for your container process.

For example, capabilities such as below are usually not needed for container process:

```
NET_ADMIN
SYS_ADMIN
SYS_MODULE
```

Audit:

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

Verify that the added and dropped Linux Kernel Capabilities are in line with the ones needed for container process for each container instance.

Remediation:

Execute the below command to add needed capabilities:

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

For example,

```
docker run --interactive --tty --cap-add={"NET_ADMIN","SYS_ADMIN"} centos:latest
/bin/bash
```

Execute the below command to drop unneeded capabilities:

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

For example,

```
docker run --interactive --tty --cap-drop={"SETUID","SETGID"} centos:latest /bin/bash
```

Alternatively,

You may choose to drop all capabilities and add only add the needed ones:

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

For example,

```
docker run --interactive --tty --cap-drop=all --cap-add={"NET_ADMIN","SYS_ADMIN"}
centos:latest /bin/bash
```

Impact:

Based on what Linux Kernel Capabilities were added or dropped, restrictions within the container would apply.

Default Value:

By default, below capabilities are available for 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/articles/security/#linux-kernel-capabilities
2. https://github.com/docker/docker/blob/master/daemon/execdriver/native/template/default_template.go
3. http://man7.org/linux/man-pages/man7/capabilities.7.html
4. http://www.oreilly.com/webops-perf/free/files/docker-security.pdf

5.4 Do not use privileged containers (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Using the `--privileged` flag gives all Linux Kernel Capabilities to the container thus overwriting the `--cap-add` and `--cap-drop` flags. Ensure that it is not used.

Rationale:

The `--privileged` flag gives all capabilities to the container, and it also lifts all the limitations enforced by the device cgroup controller. In other words, the container can then do almost everything that the host can do. This flag exists to allow special use-cases, like running Docker within Docker.

Audit:

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

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

Remediation:

Do not run container with the `--privileged` flag.

For example, do not start a container as below:

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

Impact:

Linux Kernel Capabilities other than defaults would not be available for use within container.

Default Value:

False.

References:

1. https://docs.docker.com/reference/commandline/cli

5.5 Do not mount sensitive host system directories on containers (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Sensitive host system directories such as below should not be allowed 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 would be possible to make changes to files within those sensitive directories. The changes might bring down security implications or unwarranted changes that could put the Docker host in compromised state.

Audit:

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

The above commands would return the list of current mapped directories and whether they are mounted in read-write mode for each container instance.

Remediation:

Do not mount host sensitive directories on containers especially in read-write mode.

Impact:

None.

Default Value:

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

References:

1. https://docs.docker.com/userguide/dockervolumes

5.6 Do not run ssh within containers (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

SSH server should not be running within the container. You should SSH into the Docker host, and use nsenter tool to enter a container from a remote host.

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 SSH server from the container and use `nsenter` or any other commands such as `docker exec` or `docker attach` to interact with the container instance.

```
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/

5.7 Do not map privileged ports within containers (Scored)

Profile Applicability:

- Level 1 - Docker

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 allows a container port to be mapped to a privileged port.

Rationale:

By default, if the user does not specifically declare the container port to host port mapping, Docker automatically and correctly maps the container port to one available in `49153-65535` block on the host. But, Docker allows a container port to be mapped to a privileged port on the host if the user explicitly declared it. This is so because containers are executed with `NET_BIND_SERVICE` Linux kernel capability that does not restrict the privileged port mapping. The privileged ports receive and transmit various sensitive and privileged data. Allowing containers to use them can bring serious implications.

Audit:

List all running containers instances and their port mapping by executing the below command:

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

Review the list and ensure that container ports are not mapped to host port numbers below `1024`.

Remediation:

Do not map the container ports to privileged host ports when starting a container. 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. http://docs.docker.com/articles/networking/#binding-ports
2. https://www.adayinthelifeof.nl/2012/03/12/why-putting-ssh-on-another-port-than-22-is-bad-idea

5.8 Open only needed ports on container (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Dockerfile for a container image defines the ports to be opened by default on a container instance. The list of ports may or may not be relevant to the application you are running within the container.

Rationale:

A container can be run just with the ports defined in the Dockerfile for its image or can be arbitrarily passed run time parameters to open a list of ports. Additionally, Overtime, Dockerfile may undergo various changes and the list of exposed ports may or may not be relevant to the application you are running within the container. Opening unneeded ports increase the attack surface of the container and the containerized application. As a recommended practice, do not open unneeded ports.

Audit:

List all the running instances of containers and their port mapping by executing the below command:

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

Review the list and ensure that the ports mapped are the ones that are really needed for the container.

Remediation:

Fix the Dockerfile of the container image to expose only needed ports by your containerized application. You can also completely ignore the list of ports defined in the Dockerfile by **NOT** using '`-P`' (UPPERCASE) or '`--publish-all`' flag when starting the container. 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 `EXPOSE` instruction for an image are opened when container is run with '`-P`' or '`--publish-all`' flag.

References:

1. https://docs.docker.com/articles/networking/#binding-ports

5.9 Do not share the host's network namespace (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

The networking mode on a container when set to '`--net=host`', skips placing the container inside separate network stack. In essence, this choice tells Docker to not containerize the container's networking. This would network-wise mean that the container lives "outside" in the main Docker host and has full access to its network interfaces.

Rationale:

This is potentially dangerous. It allows the container process to open low-numbered ports like any other `root` process. It also allows the container to access network services like D-bus on the Docker host. Thus, a container process can potentially do unexpected things such as shutting down the Docker host. You should not use this option.

Audit:

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

If the above command returns '`NetworkMode=host`', it means '`--net=host`' option was passed when container was started. This would be non-compliant. It should return `bridge`, `none`, or `container:$Container_Instance` to be compliant.

Remediation:

Do not pass '`--net=host`' option when starting the container.

Impact:

None.

Default Value:

By default, container connects to Docker `bridge`.

References:

1. http://docs.docker.com/articles/networking/#how-docker-networks-a-container
2. https://github.com/docker/docker/issues/6401

5.10 Limit memory usage for container (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

By default, all containers on a Docker host share the resources equally. By using the resource management capabilities of Docker host, such as memory limit, you can control the amount of memory that a container may consume.

Rationale:

By default, container can use all of the memory on the host. You can use memory limit mechanism to prevent a denial of service arising from one container consuming all of the host's resources such that other containers on the same host cannot perform their intended functions. Having no limit on memory can lead to issues where one container can easily make the whole system unstable and as a result unusable.

Audit:

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

If the above command returns `0`, it means the memory limits are not in place. If the above command returns a non-zero value, it means memory limits are in place.

Remediation:

Run the container with only as much memory as required. Always run the container using the `'--memory'` argument. You should start the container as below:

$> docker run <Run arguments> --memory <memory-size> <Container Image Name or ID> <Command>

For example,

```
docker run --interactive --tty --memory 256m centos /bin/bash
```

In the above example, the container is started with a memory limit of 256 MB.

Note: Please note that the output of the below command would return 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 the above container instance, the output of the above command would be `2.68435456e+08` and NOT 256m. You should convert this value using a scientific calculator or programmatic methods.

Impact:

If you do not set proper limits, the container process may have to starve.

Default Value:

By default, all containers on a Docker host share the resources equally. No memory limits are enforced.

References:

1. https://goldmann.pl/blog/2014/09/11/resource-management-in-docker/
2. http://docs.docker.com/reference/commandline/cli/#run
3. https://docs.docker.com/articles/runmetrics/

5.11 Set container CPU priority appropriately (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

By default, all containers on a Docker host share the resources equally. By using the resource management capabilities of Docker host, such as CPU shares, you can control the host CPU resources that a container may consume.

Rationale:

By default, CPU time is divided between containers equally. If it is desired, to control the CPU time amongst the container instances, you can use CPU sharing feature. CPU sharing allows to prioritize one container over the other and forbids the lower priority container to claim CPU resources more often. This ensures that the high priority containers are served better.

Audit:

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

If the above command returns `0` or `1024`, it means the CPU shares are not in place. If the above command returns a non-zero value other than `1024`, it means CPU shares are in place.

Remediation:

Manage the CPU shares between your containers. To do so start the container using the `'--cpu-shares'` argument. You may start the container as below:

$> docker run <Run arguments> --cpu-shares <CPU shares> <Container Image Name or ID> <Command>

For example,

```
docker run --interactive --tty --cpu-shares 512 centos /bin/bash
```

In the above example, the container is started with a CPU shares of 50% of what the other containers use. So, if the other container has CPU shares of 80%, this container will have CPU shares of 40%.

Note: 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 `/sys/fs/cgroup/cpu/system.slice/` directory.
2. Check your container instance ID using '`docker ps`' command.
3. Now, inside the above directory (in step 1), you would have a directory by name '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. So, even if there are no CPU shares configured using '`-c`' or '`--cpu-shares`' argument in the '`docker run`' command, this file will have a value of '`1024`'.

If we set one container's CPU shares to `512` it will receive half of the CPU time compared to the other container. So, take `1024` as 100% and then do quick math to derive the number that you should set for respective CPU shares. For example, use `512` if you want to set 50% and 256 if you want to set 25%.

Impact:

If you do not set proper CPU shares, the container process may have to starve if the resources on the host are not available. If the CPU resources on the host are free, CPU shares do not place any restrictions on the CPU that the container may use.

Default Value:

By default, all containers on a Docker host share the resources equally. No CPU shares are enforced.

References:

1. https://goldmann.pl/blog/2014/09/11/resource-management-in-docker/
2. http://docs.docker.com/reference/commandline/cli/#run
3. https://docs.docker.com/articles/runmetrics/

5.12 Mount container's root filesystem as read only (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

The container's `root` file system should be treated as a 'golden image' and any writes to the `root` filesystem should be avoided. You should explicitly define a container volume for writing.

Rationale:

You should not be writing data within containers. The data volume belonging to a container should be explicitly defined and administered. This is useful in many cases where the admin controls where they would want developers to write files and errors. Also, this has other advantages such as below:

- This leads to an immutable infrastructure
- Since the container instance cannot be written to, there is no need to audit instance divergence
- Reduced security attack vectors since the instance cannot be tampered with or written to
- Ability to use a purely volume based backup without backing up anything from the instance

Audit:

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

If the above command returns '`true`', it means the `root` filesystem is mounted read-only. If the above command returns '`false`', it means the `root` filesystem is writable.

Additionally, you may use below command to find the differences between the container instance and its corresponding image.

```
docker diff $INSTANCE_ID
```

Remediation:

Add a '`--read-only`' flag to allow the container's `root` filesystem to be mounted as read only. This can be used in combination with volumes to force a container's process to only write to locations that will be persisted.

You should run the container as below:

$> docker run <Run arguments> --read-only -v <writable-volume> <Container Image Name or ID> <Command>

For example,

```
docker run --interactive --tty --read-only --volume /centdata centos /bin/bash
```

This would run the container with read-only `root` filesystem and would use '`centdata`' as container volume for writing.

Impact:

The container root file system would not be writable. You should explicitly define a volume for the container for writing.

Default Value:

By default, a container will have its `root` filesystem writable allowing processes to write files anywhere.

References:

1. http://docs.docker.com/reference/commandline/cli/#run

5.13 Bind incoming container traffic to a specific host interface (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

By default, Docker containers can make connections to the outside world, but the outside world cannot connect to containers. Each outgoing connection will appear to originate from one of the host machine's own IP addresses. 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 the exposed ports on any network interface. This might not be desired and may not be secured. Many a times a particular interface is exposed externally and services such as intrusion detection, intrusion prevention, firewall, load balancing, etc. are run on those interfaces to screen incoming public traffic. Hence, you should not accept incoming connections on any interface. You should only allow incoming connections from a particular external interface.

Audit:

List all the running instances of containers and their port mapping by executing the below command:

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

Review the list and ensure that the exposed container ports are tied to a particular interface and not to the wild card IP address - '`0.0.0.0`'.

For example,

If the above command returns as below, then 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 tied to a particular interface on the host as below, then this recommendation is configured as desired and is compliant.

```
Ports=map[443/tcp:<nil> 80/tcp:[map[HostIp:10.2.3.4 HostPort:49153]]]
```

Remediation:

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 `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/articles/networking/#binding-container-ports-to-the-host

5.14 Set the 'on-failure' container restart policy to 5 (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Using the '`--restart`' flag in '`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. Thus, it is recommended to use '`on-failure`' restart policy and limit it to maximum of `5` restart attempts.

Audit:

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
RestartPolicyName={{ .HostConfig.RestartPolicy.Name }} MaximumRetryCount={{
.HostConfig.RestartPolicy.MaximumRetryCount }}'
```

- If the above command returns '`RestartPolicyName=always`', then the system is not configured as desired and hence this recommendation is non-compliant.
- If the above command returns '`RestartPolicyName=no`' or just '`RestartPolicyName=`', then the restart policies are not being used and the container would never be restarted of its own. This recommendation is then Not Applicable and can be assumed to be compliant.
- 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 a container is desired to be restarted of its own, then start the container as below:

$> docker run <Run arguments> --restart=on-failure:5 <Container Image Name or ID> <Command>

For example,

```
docker run --detach --restart=on-failure:5 nginx
```

Impact:

The container would attempt to restart only for `5` times.

Default Value:

By default, containers are not configured with restart policies. Hence, containers do not attempt to restart of their own.

References:

1. http://docs.docker.com/reference/commandline/cli/#restart-policies

5.15 Do not share the host's process namespace (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Process ID (PID) namespaces isolate the process ID number space, meaning that processes in different PID namespaces can have the same PID. This is process level isolation between containers and the host.

Rationale:

PID namespace provides separation of processes. The PID Namespace removes the view of the system processes, and allows process ids to be reused including PID `1`. If the host's PID namespace is shared with the container, it would basically allow processes within the container to see all of the processes on the host system. This breaks the benefit of process level isolation between the host and the containers. Someone having access to the container can eventually know all the processes running on the host system and can even kill the host system processes from within the container. This can be catastrophic. Hence, do not share the host's process namespace with the containers.

Audit:

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

If the above command returns '`host`', it means the host PID namespace is shared with the container else this recommendation is compliant.

Remediation:

Do not start a container with '`--pid=host`' argument.

For example, do not start a container as below:

```
docker run --interactive --tty --pid=host centos /bin/bash
```

Impact:

Container processes cannot see the processes on the host system. In certain cases, you want your container to share the host's process namespace. For example, you could build a container with debugging tools like `strace` or `gdb`, but want to use these tools when debugging processes within the container. If this is desired, then share only one (or needed) host process by 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 host's process namespace is not shared with the containers.

References:

1. https://docs.docker.com/reference/run/#pid-settings
2. http://man7.org/linux/man-pages/man7/pid_namespaces.7.html

5.16 Do not share the host's IPC namespace (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

IPC (POSIX/SysV IPC) namespace provides separation of named shared memory segments, semaphores and message queues. IPC namespace on the host thus should not be shared with the containers and should remain isolated.

Rationale:

IPC namespace provides separation of IPC between the host and containers. If the host's IPC namespace is shared with the container, it would basically allow processes within the container to see all of the IPC on the host system. This breaks the benefit of IPC level isolation between the host and the containers. Someone having access to the container can eventually manipulate the host IPC. This can be catastrophic. Hence, do not share the host's IPC namespace with the containers.

Audit:

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

If the above command returns '`host`', it means the host IPC namespace is shared with the container. If the above command returns nothing, then the host's IPC namespace is not shared. This recommendation is then compliant.

Remediation:

Do not start a container with '`--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 to accelerate inter-process communication. It is commonly used by high performance applications. If such applications are containerized into multiple containers, you might need to share the IPC namespace of the containers to achieve high performance. In such cases, you should still be sharing container specific IPC

namespaces only and not the host IPC namespace. You may share the container's IPC namespace with other container as below:

$> docker run <Run arguments> --ipc=container:$INSTANCE_ID <Container Image Name or ID> <Command>

For example,

```
docker run --interactive --tty --ipc=container:e3a7a1a97c58 centos /bin/bash
```

Default Value:

By default, all containers have the IPC namespace enabled and host IPC namespace is not shared with any container.

References:

1. https://docs.docker.com/reference/run/#ipc-settings
2. http://man7.org/linux/man-pages/man7/namespaces.7.html

5.17 Do not directly expose host devices to containers (Not Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Host devices can be directly exposed to containers at runtime. Do not directly expose host devices to containers especially for containers that are not trusted.

Rationale:

The '`--device`' option exposes the host devices to the containers and consequently the containers can directly access such host devices. You would not require the container to run in '`privileged`' mode to access and manipulate the host devices. By default, the container will be able to read, write and mknod these devices. Additionally, it is possible for containers to remove block devices from the host. Hence, do not expose host devices to containers directly.

If at all, you would want to expose the host device to a container, use the sharing permissions appropriately:

- r - read only
- w - writable
- m - mknod allowed

Audit:

```
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

Verify that the host device is needed to be accessed from within the container and the permissions required are correctly set. If the above command returns [], then the container does not have access to host devices. This recommendation can be assumed to be compliant.

Remediation:

Do not directly expose the host devices to containers. If at all, you need to expose the host devices to containers, use the correct set of permissions:

For example, do not start a container as below:

```
docker run --interactive --tty --device=/dev/tty0:/dev/tty0:rwm --
device=/dev/temp_sda:/dev/temp_sda:rwm centos bash
```

For example, share the host device with correct 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 the host devices directly within the containers.

Default Value:

By default, no host devices are exposed to containers. If you do not provide sharing permissions and choose to expose a host device to a container, the host device would be exposed with `read`, `write` and `mknod` permissions.

References:

1. http://docs.docker.com/reference/commandline/cli/#run

5.18 Override default ulimit at runtime only if needed (Not Scored)

Profile Applicability:

- Level 1 - Docker

Description:

The default ulimit is set at the Docker daemon level. However, you may override the default ulimit setting, if needed, during container runtime.

Rationale:

`ulimit` provides control over the resources available to the shell and to processes started by it. Setting system resource limits judiciously saves you from many disasters such as a fork bomb. Sometimes, even friendly users and legitimate processes can overuse system resources and in-turn can make the system unusable.

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, do not make this a practice. If most of the container instances are overriding default ulimit settings, consider changing the default ulimit settings to something that is appropriate for your needs.

Audit:

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

The above command should return `Ulimits=<no value>` for each container instance until and unless there is an exception and a need to override the default ulimit settings.

Remediation:

Only override the default ulimit settings if needed.

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 the ulimits are not set properly, the desired resource control might not be achieved and might even make the system unusable.

Default Value:

Container instances inherit the default ulimit settings set at the Docker daemon level.

References:

1. http://docs.docker.com/reference/commandline/cli/#setting-ulimits-in-a-container
2. Command: man setrlimit
3. http://www.oreilly.com/webops-perf/free/files/docker-security.pdf

Notes:

There are multiple options within ulimit that you can use. Few important ulimits that you can control are cpu (CPU time limit in seconds), fsize (maximum size of a file process can create), memlock (max number of bytes of memory that maybe locked into RAM), nproc (mx number of processes) etc. Full list can be seen from the command "man setrlimit"

5.19 Do not set mount propagation mode to shared (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Mount propagation mode allows mounting volumes in shared, slave or private mode on a container. Do not use shared mount propagation mode until needed.

Rationale:

A shared mount is replicated at all mounts and the changes made at any mount point are propagated to all mounts. Mounting a volume in shared mode does not restrict any other container to mount and make changes to that volume. This might be catastrophic if the mounted volume is sensitive to changes. Do not set mount propagation mode to shared until needed.

Audit:

```
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
Propagation={{range $mnt := .Mounts}} {{json $mnt.Propagation}} {{end}}'
```

The above command would return the propagation mode for mounted volumes. 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 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://github.com/docker/docker/pull/17034
2. https://docs.docker.com/engine/reference/run/
3. https://www.kernel.org/doc/Documentation/filesystems/sharedsubtree.txt

5.20 Do not share the host's UTS namespace (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

UTS namespaces provide isolation of two system identifiers: the hostname and the NIS domain name. It is used for setting the hostname and the domain that is visible to running processes in that namespace. Processes running within containers do not typically require to know hostname and domain name. Hence, the namespace should not be shared with the host.

Rationale:

Sharing the UTS namespace with the host provides full permission to the container to change the hostname of the host. This is insecure and should not be allowed.

Audit:

```
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:

Do not start a container with '`--uts=host`' argument.

For example, do not start a container as below:

```
docker run --rm --interactive --tty --uts=host rhel7.2
```

Impact:

None.

Default Value:

By default, all containers have the UTS namespace enabled and host UTS namespace is not shared with any container.

References:

1. https://docs.docker.com/engine/reference/run/
2. http://man7.org/linux/man-pages/man7/namespaces.7.html

5.21 Do not disable default seccomp profile (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Seccomp filtering provides a means for a process to specify a filter for incoming system calls. The default Docker seccomp profile works on whitelist basis and allows 311 system calls blocking all others. It should not be disabled unless it hinders 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 the applications do not need all the system calls and thus benefit by having a reduced set of available system calls. The reduced set of system calls reduces the total kernel surface exposed to the application and thus improvises application security.

Audit:

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

The above command should return "`<no value>`" or your modified seccomp profile. If it returns `[seccomp:unconfined]`, that means this recommendation is non-compliant and the container is running without any seccomp profiles.

Remediation:

By default, seccomp profiles are enabled. You do not need to do anything unless you want to modify and use the 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/
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

5.22 Do not docker exec commands with privileged option (Scored)

Profile Applicability:

- Level 2 - Docker

Description:

Do not `docker exec` with `--privileged` option.

Rationale:

Using `--privileged` option in `docker exec` gives extended Linux capabilities to the command. This could potentially be insecure and unsafe to do especially when you are running containers with dropped capabilities or with enhanced restrictions.

Audit:

If you have auditing enabled as prescribed in Section 1, you can use the below command to filter out `docker exec` commands that used `--privileged` option.

```
ausearch -k docker | grep exec | grep privileged
```

Remediation:

Do not use `--privileged` option in `docker exec` command.

Impact:

None. If you need enhanced capabilities within the container, then run the container with the needed capabilities.

Default Value:

By default, `docker exec` command runs without `--privileged` option.

References:

1. https://docs.docker.com/engine/reference/commandline/exec/

5.23 Do not docker exec commands with user option (Scored)

Profile Applicability:

- Level 2 - Docker

Description:

Do not `docker exec` with `--user` option.

Rationale:

Using `--user` option in `docker exec` executes the command within the container as that user. This could potentially be insecure and unsafe to do especially when you are running containers with dropped capabilities or with enhanced restrictions.

For example, suppose your container is running as tomcat user (or any other non-root user), it would be possible to run a command through `docker exec` as `root` with `--user=root` option. This could potentially be dangerous.

Audit:

If you have auditing enabled as prescribed in Section 1, you can use the below command to filter out `docker exec` commands that used `--user` option.

```
ausearch -k docker | grep exec | grep user
```

Remediation:

Do not use **`--user`** option in `docker exec` command.

Impact:

None.

Default Value:

By default, `docker exec` command runs without `--user` option.

References:

1. https://docs.docker.com/engine/reference/commandline/exec/

5.24 Confirm cgroup usage (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

It is possible to attach to a particular cgroup on container run. Confirming cgroup usage would ensure that containers are running under defined cgroups.

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.

At run-time, it is possible to attach to a different cgroup other than the one that was expected to be used. This usage should be monitored and confirmed. By attaching to a different cgroup than the one that is expected, excess permissions and resources might be granted to the container and thus, can prove to be unsafe.

Audit:

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

The above command would return the cgroup under which the containers are running. If it is blank, it means containers are running under default docker cgroup. In that case, this recommendation is compliant. If the containers are found to be running under cgroup other than the one that was expected, this recommendation is non-compliant.

Remediation:

Do not use `--cgroup-parent` option in `docker run` command unless needed.

Impact:

None.

Default Value:

By default, containers run under `docker` cgroup.

References:

1. https://docs.docker.com/engine/reference/run/#specifying-custom-cgroups
2. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/ch01.html

5.25 Restrict container from acquiring additional privileges (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

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. It persists across fork, clone and execve. The `no_new_priv` bit ensures that the process or its children processes do not gain any additional privileges via suid or sgid bits. This way a lot of dangerous operations become a lot less dangerous because there is no possibility of subverting privileged binaries.

Audit:

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

The above command should return all the security options currently configured for the containers. `no-new-privileges` should also be one of them.

Remediation:

Start a container as below:

docker run <run-options> --security-opt=no-new-privileges <IMAGE> <CMD>

For example,

```
docker run --rm -it --security-opt=no-new-privileges ubuntu bash
```

Impact:

`no_new_priv` prevents LSMs like SELinux from transitioning to process labels that have access not allowed to the current process.

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/

5.26 Check container health at runtime (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

If the container image does not have an `HEALTHCHECK` instruction defined, use `--health-cmd` parameter at container runtime for checking container health.

Rationale:

One of the important security triads is availability. 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, you could take necessary actions.

Audit:

Run the below command and ensure that all the containers are reporting health status:

```
docker ps --quiet | xargs docker inspect --format '{{ .Id }}: Health={{
.State.Health.Status }}'
```

Remediation:

Run the container using `--health-cmd` and the other parameters.

For example,

```
docker run -d --health-cmd='stat /etc/passwd || exit 1' nginx
```

Impact:

None.

Default Value:

By default, health checks are not done at container runtime.

References:

1. https://github.com/docker/docker/pull/22719

5.27 Ensure docker commands always get the latest version of the image (Not Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Always ensure that you are using the latest version of the image within your repository and not the cached older versions.

Rationale:

Multiple docker commands such as `docker pull`, `docker run`, etc. are known to have an issue that 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 and vulnerable images.

Audit:

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 with the latest version reported in your repository which tells if you are running the cached version or the latest copy.

Remediation:

Use proper version pinning mechanisms (the latest tag which is assigned by default is still vulnerable to caching attacks) to avoid extracting the cached older versions. Version pinning mechanisms should be used for base images, packages, and entire images too. You can customize version pinning rules as per 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

5.28 Use PIDs cgroup limit (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Use `--pids-limit` flag at container runtime.

Rationale:

Attackers could launch a fork bomb with a single command inside the container. This fork bomb can crash the entire system and requires a restart of the host to make the system functional again. PIDs cgroup `--pids-limit` will prevent this kind of attacks by restricting the number of forks that can happen inside a container at a given time.

Audit:

Run the below command 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 inside the container concurrently.

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

Remediation:

Use `--pids-limit` flag while launching the container with an appropriate value.

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 the containers unusable.

Default Value:

The Default value for `--pids-limit` is `0` which means there is no restriction on the number of forks. Also, note that PIDs cgroup limit works only for the kernel versions 4.3+.

References:

1. https://github.com/docker/docker/pull/18697
2. https://docs.docker.com/engine/reference/commandline/run/

5.29 Do not use Docker's default bridge docker0 (Not Scored)

Profile Applicability:

- Level 2 - Docker

Description:

Do not use Docker's default bridge `docker0`. Use docker's user-defined networks for container networking.

Rationale:

Docker connects virtual interfaces created in the bridge mode to a common bridge called `docker0`. This default networking model is vulnerable to ARP spoofing and MAC flooding attacks since there is no filtering applied.

Audit:

Run the below command, and verify that containers are on a user-defined network and not the default `docker0` bridge.

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

Remediation:

Follow Docker documentation and setup a user-defined network. Run all the containers in the defined network.

Impact:

You have to manage the user-defined networks.

Default Value:

By default, docker runs containers on its `docker0` bridge.

References:

1. https://github.com/nyantec/narwhal
2. https://arxiv.org/pdf/1501.02967
3. https://docs.docker.com/engine/userguide/networking/dockernetworks/

5.30 Do not share the host's user namespaces (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Do not share the host's user namespaces with the containers.

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 thus does not isolate users on the host with users on the containers.

Audit:

Run the below command and ensure that it does not return any value for `UsernsMode`. If it returns a value of `host`, it means the host user namespace is shared with the containers.

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

Remediation:

Do not share user namespaces between host and containers.

Impact:

None

Default Value:

By default, the host user namespace is shared with the containers until user namespace support is enabled.

References:

1. https://docs.docker.com/engine/reference/commandline/run/#/run
2. https://events.linuxfoundation.org/sites/events/files/slides/User%20Namespaces%20-%20ContainerCon%202015%20-%2016-9-final_0.pdf
3. https://github.com/docker/docker/pull/12648

5.31 Do not mount the Docker socket inside any containers (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

The docker socket (`docker.sock`) should not be mounted inside a container.

Rationale:

If the docker socket is mounted inside a container it would allow processes running within the container to execute docker commands which effectively allows for full control of the host.

Audit:

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

The above command would return any instances where `docker.sock` had been mapped to a container as a volume.

Remediation:

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

6 Docker Security Operations

This sections covers some of the operational security aspects for Docker deployments. These are best practices that should be followed. Most of the recommendations here are just reminders that organizations should extend their current security best practices and policies to include containers.

6.1 Perform regular security audits of your host system and containers (Not Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Perform regular security audits of your host system and containers to identify any mis-configurations or vulnerabilities that could expose your system to compromise.

Rationale:

Performing regular and dedicated security audits of your host systems and containers could provide deep security insights that you might not know in your daily course of business. The identified security weaknesses should be then mitigated and this overall improves security posture of your environment.

Audit:

Follow your organization's security audit policies and requirements.

Remediation:

Follow your organization's security audit policies and requirements.

Impact:

None.

Default Value:

Not applicable.

References:

1. http://searchsecurity.techtarget.com/IT-security-auditing-Best-practices-for-conducting-audits

6.2 Monitor Docker containers usage, performance and metering (Not Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Containers might run services that are critical for your business. Monitoring their usage, performance and metering would be of paramount importance.

Rationale:

Tracking container usage, performance and having some sort of metering around them would be important as you embrace the containers to run critical services for your business. This would give you

- Capacity Management and Optimization
- Performance Management
- Comprehensive Visibility

Such a deep visibility of container performance would help you ensure high availability of containers and minimum downtime.

Audit:

```
docker stats $(docker ps --quiet)
```

The above commands would return CPU, memory and network statistics for every running container.

Remediation:

Use a software or a container for tracking container usage, reporting performance and metering.

Impact:

To get container metrics, you would have to utilize another container in privileged mode or a software that can enter namespace of various containers. Giving unrestricted access to namespaces of all the containers might be too risky.

Default Value:

By default, for each container, runtime metrics about CPU, memory, and block I/O usage is tracked by the system via enforcement of control groups (cgroups) as below:

CPU - `/sys/fs/cgroup/cpu/system.slice/docker-$INSTANCE_ID.scope/`
Memory - `/sys/fs/cgroup/memory/system.slice/docker-$INSTANCE_ID.scope/`
Block I/O - `/sys/fs/cgroup/blkio/system.slice/docker-$INSTANCE_ID.scope/`

References:

1. https://docs.docker.com/articles/runmetrics/
2. https://github.com/google/cadvisor
3. https://docs.docker.com/reference/commandline/cli/#stats

6.3 Backup container data (Not Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Take regular backups of your container data volumes.

Rationale:

Containers might run services that are critical for your business. Taking regular data backups would ensure that if there is ever any loss of data you would still have your data in backup. The loss of data could be devastating for your business.

Audit:

Ask the system administrator whether container data volumes are regularly backed up. Verify a copy of the backup and ensure that the organization's backup policy is followed.

Additionally, you can execute the below command for each container instance to list the changed files and directories in the container's filesystem. Ideally, nothing should be stored on container's filesystem.

```
docker diff $INSTANCE_ID
```

Remediation:

You should follow your organization's policy for data backup. You can take backup of your container data volume using '`--volumes-from`' parameter as below:

$> docker run <Run arguments> --volumes-from $INSTANCE_ID -v [host-dir]:[container-dir] <Container Image Name or ID> <Command>

For example,

```
docker run --volumes-from 699ee3233b96 -v /mybackup:/backup centos tar cvf
/backup/backup.tar /exampledatatobackup
```

Impact:

None.

Default Value:

By default, no data backup happens for container data volumes.

References:

1. http://docs.docker.com/userguide/dockervolumes/#backup-restore-or-migrate-data-volumes
2. http://stackoverflow.com/questions/26331651/back-up-docker-container-that-has-a-volume
3. http://docs.docker.com/reference/commandline/cli/#diff

6.4 Avoid image sprawl (Not Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Do not keep a large number of container images on the same host. Use only tagged images as appropriate.

Rationale:

Tagged images are useful to fall back from "latest" to a specific version of an image in production. Images with unused or old tags may contain vulnerabilities that might be exploited, if instantiated. Additionally, if you fail to remove unused images from the system and there are various such redundant and unused images, the host filesystem may become full and could lead to denial of service.

Audit:

Step 1 Make a list of all image IDs that are currently instantiated by executing below command:

```
docker images --quiet | xargs docker inspect --format '{{ .Id }}: Image={{
.Config.Image }}'
```

Step 2: List all the images present on the system by executing below command:

```
docker images
```

Step 3: Compare the list of image IDs populated from Step 1 and Step 2 and find out images that are currently not being instantiated. If any such unused or old images are found, discuss with the system administrator the need to keep such images on the system. If such a need is not justified enough, then this recommendation is non-compliant.

Remediation:

Keep the set of the images that you actually need and establish a workflow to remove old or stale images from the host. Additionally, use features such as pull-by-digest to get specific images from the registry.

Additionally, you can follow below set of steps to find out unused images on the system and delete them.

Step 1 Make a list of all image IDs that are currently instantiated by executing below command:

```
docker images --quiet | xargs docker inspect --format '{{ .Id }}: Image={{
.Config.Image }}'
```

Step 2: List all the images present on the system by executing below command:

```
docker images
```

Step 3: Compare the list of image IDs populated from Step 1 and Step 2 and find out images that are currently not being instantiated.

Step 4: Decide if you want to keep the images that are not currently in use. If not delete them by executing below command:

```
docker rmi $IMAGE_ID
```

Impact:

None

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. http://docs.docker.com/reference/commandline/cli/#rmi
5. http://docs.docker.com/reference/commandline/cli/#pull
6. https://github.com/docker/docker/pull/11109

6.5 Avoid container sprawl (Not Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Do 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 indirectly leads to Docker images that 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 the manageable number of containers on a particular host makes the situation vulnerable to mishandling, misconfiguration and fragmentation. Thus, avoid container sprawl and keep the number of containers on a host to a manageable total.

Audit:

Execute `docker info` to find the total number of containers you have on the host:

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

Now, execute the below commands 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 on the host is large (say 25 or more), then perhaps, the containers are sprawled on the host.

Remediation:

Periodically check your container inventory per host and clean up the stopped containers using the below command:

```
docker container prune
```

Impact:

If you keep way too few number of containers per host, then perhaps you are not utilizing your host resources very adequately.

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

Appendix: Summary Table

Control		Set Correctly	
		Yes	No
1	**Host Configuration**		
1.1	Create a separate partition for containers (Scored)	☐	☐
1.2	Harden the container host (Not Scored)	☐	☐
1.3	Keep Docker up to date (Not Scored)	☐	☐
1.4	Only allow trusted users to control Docker daemon (Scored)	☐	☐
1.5	Audit docker daemon (Scored)	☐	☐
1.6	Audit Docker files and directories - /var/lib/docker (Scored)	☐	☐
1.7	Audit Docker files and directories - /etc/docker (Scored)	☐	☐
1.8	Audit Docker files and directories - docker.service (Scored)	☐	☐
1.9	Audit Docker files and directories - docker.socket (Scored)	☐	☐
1.10	Audit Docker files and directories - /etc/default/docker (Scored)	☐	☐
1.11	Audit Docker files and directories - /etc/docker/daemon.json (Scored)	☐	☐
1.12	Audit Docker files and directories - /usr/bin/docker-containerd (Scored)	☐	☐
1.13	Audit Docker files and directories - /usr/bin/docker-runc (Scored)	☐	☐
2	**Docker daemon configuration**		
2.1	Restrict network traffic between containers (Scored)	☐	☐
2.2	Set the logging level (Scored)	☐	☐
2.3	Allow Docker to make changes to iptables (Scored)	☐	☐
2.4	Do not use insecure registries (Scored)	☐	☐
2.5	Do not use the aufs storage driver (Scored)	☐	☐
2.6	Configure TLS authentication for Docker daemon (Scored)	☐	☐
2.7	Set default ulimit as appropriate (Not Scored)	☐	☐
2.8	Enable user namespace support (Scored)	☐	☐
2.9	Confirm default cgroup usage (Scored)	☐	☐
2.10	Do not change base device size until needed (Scored)	☐	☐
2.11	Use authorization plugin (Scored)	☐	☐
2.12	Configure centralized and remote logging (Scored)	☐	☐
2.13	Disable operations on legacy registry (v1) (Scored)	☐	☐
2.14	Enable live restore (Scored)	☐	☐
2.15	Do not enable swarm mode, if not needed (Scored)	☐	☐
2.16	Control the number of manager nodes in a swarm (Scored)	☐	☐
2.17	Bind swarm services to a specific host interface (Scored)	☐	☐

2.18	Disable Userland Proxy (Scored)	☐	☐
2.19	Encrypt data exchanged between containers on different nodes on the overlay network (Scored)	☐	☐
2.20	Apply a daemon-wide custom seccomp profile, if needed (Not Scored)	☐	☐
2.21	Avoid experimental features in production (Scored)	☐	☐
2.22	Use Docker's secret management commands for managing secrets in a Swarm cluster (Not Scored)	☐	☐
2.23	Run swarm manager in auto-lock mode (Scored)	☐	☐
2.24	Rotate swarm manager auto-lock key periodically (Not Scored)	☐	☐
3	**Docker daemon configuration files**		
3.1	Verify that docker.service file ownership is set to root:root (Scored)	☐	☐
3.2	Verify that docker.service file permissions are set to 644 or more restrictive (Scored)	☐	☐
3.3	Verify that docker.socket file ownership is set to root:root (Scored)	☐	☐
3.4	Verify that docker.socket file permissions are set to 644 or more restrictive (Scored)	☐	☐
3.5	Verify that /etc/docker directory ownership is set to root:root (Scored)	☐	☐
3.6	Verify that /etc/docker directory permissions are set to 755 or more restrictive (Scored)	☐	☐
3.7	Verify that registry certificate file ownership is set to root:root (Scored)	☐	☐
3.8	Verify that registry certificate file permissions are set to 444 or more restrictive (Scored)	☐	☐
3.9	Verify that TLS CA certificate file ownership is set to root:root (Scored)	☐	☐
3.10	Verify that TLS CA certificate file permissions are set to 444 or more restrictive (Scored)	☐	☐
3.11	Verify that Docker server certificate file ownership is set to root:root (Scored)	☐	☐
3.12	Verify that Docker server certificate file permissions are set to 444 or more restrictive (Scored)	☐	☐
3.13	Verify that Docker server certificate key file ownership is set to root:root (Scored)	☐	☐
3.14	Verify that Docker server certificate key file permissions are set to 400 (Scored)	☐	☐
3.15	Verify that Docker socket file ownership is set to root:docker (Scored)	☐	☐
3.16	Verify that Docker socket file permissions are set to 660 or more restrictive (Scored)	☐	☐

3.17	Verify that daemon.json file ownership is set to root:root (Scored)	☐	☐
3.18	Verify that daemon.json file permissions are set to 644 or more restrictive (Scored)	☐	☐
3.19	Verify that /etc/default/docker file ownership is set to root:root (Scored)	☐	☐
3.20	Verify that /etc/default/docker file permissions are set to 644 or more restrictive (Scored)	☐	☐
4	**Container Images and Build File**		
4.1	Create a user for the container (Scored)	☐	☐
4.2	Use trusted base images for containers (Not Scored)	☐	☐
4.3	Do not install unnecessary packages in the container (Not Scored)	☐	☐
4.4	Scan and rebuild the images to include security patches (Not Scored)	☐	☐
4.5	Enable Content trust for Docker (Scored)	☐	☐
4.6	Add HEALTHCHECK instruction to the container image (Scored)	☐	☐
4.7	Do not use update instructions alone in the Dockerfile (Not Scored)	☐	☐
4.8	Remove setuid and setgid permissions in the images (Not Scored)	☐	☐
4.9	Use COPY instead of ADD in Dockerfile (Not Scored)	☐	☐
4.10	Do not store secrets in Dockerfiles (Not Scored)	☐	☐
4.11	Install verified packages only (Not Scored)	☐	☐
5	**Container Runtime**		
5.1	Do not disable AppArmor Profile (Scored)	☐	☐
5.2	Verify SELinux security options, if applicable (Scored)	☐	☐
5.3	Restrict Linux Kernel Capabilities within containers (Scored)	☐	☐
5.4	Do not use privileged containers (Scored)	☐	☐
5.5	Do not mount sensitive host system directories on containers (Scored)	☐	☐
5.6	Do not run ssh within containers (Scored)	☐	☐
5.7	Do not map privileged ports within containers (Scored)	☐	☐
5.8	Open only needed ports on container (Scored)	☐	☐
5.9	Do not share the host's network namespace (Scored)	☐	☐
5.10	Limit memory usage for container (Scored)	☐	☐
5.11	Set container CPU priority appropriately (Scored)	☐	☐
5.12	Mount container's root filesystem as read only (Scored)	☐	☐
5.13	Bind incoming container traffic to a specific host interface (Scored)	☐	☐
5.14	Set the 'on-failure' container restart policy to 5 (Scored)	☐	☐

5.15	Do not share the host's process namespace (Scored)	☐	☐
5.16	Do not share the host's IPC namespace (Scored)	☐	☐
5.17	Do not directly expose host devices to containers (Not Scored)	☐	☐
5.18	Override default ulimit at runtime only if needed (Not Scored)	☐	☐
5.19	Do not set mount propagation mode to shared (Scored)	☐	☐
5.20	Do not share the host's UTS namespace (Scored)	☐	☐
5.21	Do not disable default seccomp profile (Scored)	☐	☐
5.22	Do not docker exec commands with privileged option (Scored)	☐	☐
5.23	Do not docker exec commands with user option (Scored)	☐	☐
5.24	Confirm cgroup usage (Scored)	☐	☐
5.25	Restrict container from acquiring additional privileges (Scored)	☐	☐
5.26	Check container health at runtime (Scored)	☐	☐
5.27	Ensure docker commands always get the latest version of the image (Not Scored)	☐	☐
5.28	Use PIDs cgroup limit (Scored)	☐	☐
5.29	Do not use Docker's default bridge docker0 (Not Scored)	☐	☐
5.30	Do not share the host's user namespaces (Scored)	☐	☐
5.31	Do not mount the Docker socket inside any containers (Scored)	☐	☐
6	**Docker Security Operations**		
6.1	Perform regular security audits of your host system and containers (Not Scored)	☐	☐
6.2	Monitor Docker containers usage, performance and metering (Not Scored)	☐	☐
6.3	Backup container data (Not Scored)	☐	☐
6.4	Avoid image sprawl (Not Scored)	☐	☐
6.5	Avoid container sprawl (Not Scored)	☐	☐

Appendix: Change History

Date	Version	Changes for this version
01-19-2017	1.0.0	Initial Release
01-19-2017	1.0.0	ADD- 2.19 Encrypt data exchanged between containers on different nodes on the overlay network
01-19-2017	1.0.0	ADD - 2.20 Apply a daemon-wide custom seccomp profile, if needed
01-19-2017	1.0.0	ADD - 2.21 Avoid experimental features in production
01-19-2017	1.0.0	ADD - 2.22 Use Docker's secret management commands for managing secrets in a Swarm cluster
01-19-2017	1.0.0	ADD - 2.23 Run swarm manager in auto-lock mode
01-19-2017	1.0.0	ADD - 2.24 Rotate swarm manager auto-lock key periodically
01-19-2017	1.0.0	UPDATES - Updated Audit Procedure - 2.8 Enable user namespace support
01-19-2017	1.0.0	UPDATES - Updated Remediation and Audit Procedure - 6.5 Avoid container sprawl
01-19-2017	1.0.0	UPDATES - Re-worded - 1.3 Keep Docker up to date
01-19-2017	1.0.0	REMOVED - 1.2 Use the updated Linux Kernel

01-19-2017	1.0.0	REMOVED - 1.3 Remove all non-essential services from the host