CENTER FOR INTERNET SECURITY

CIS Docker 1.6 Benchmark

v1.0.0 - 04-22-2015

The CIS Security Benchmarks division provides consensus-oriented information security products, services, tools, metrics, suggestions, and recommendations (the "SB Products") as a public service to Internet users worldwide. Downloading or using SB Products in any way signifies and confirms your acceptance of and your binding agreement to these CIS Security Benchmarks Terms of Use.

CIS SECURITY BENCHMARKS TERMS OF USE

BOTH CIS SECURITY BENCHMARKS DIVISION MEMBERS AND NON-MEMBERS MAY:

- Download, install, and use each of the SB Products on a single computer, and/or
- Print one or more copies of any SB Product that is in a .txt, .pdf, .doc, .mcw, or .rtf format, but only if each such copy is printed in its entirety and is kept intact, including without limitation the text of these CIS Security Benchmarks Terms of Use.

UNDER THE FOLLOWING TERMS AND CONDITIONS:

- **SB Products Provided As Is**. CIS is providing the SB Products "as is" and "as available" without: (1) any representations, warranties, or covenants of any kind whatsoever (including the absence of any warranty regarding: (a) the effect or lack of effect of any SB Product on the operation or the security of any network, system, software, hardware, or any component of any of them, and (b) the accuracy, utility, reliability, timeliness, or completeness of any SB Product); or (2) the responsibility to make or notify you of any corrections, updates, upgrades, or fixes.
- **Intellectual Property and Rights Reserved**. You are not acquiring any title or ownership rights in or to any SB Product, and full title and all ownership rights to the SB Products remain the exclusive property of CIS. All rights to the SB Products not expressly granted in these Terms of Use are hereby reserved.
- **Restrictions**. You acknowledge and agree that you may not: (1) decompile, dis-assemble, alter, reverse engineer, or otherwise attempt to derive the source code for any software SB Product that is not already in the form of source code; (2) distribute, redistribute, sell, rent, lease, sublicense or otherwise transfer or exploit any rights to any SB Product in any way or for any purpose; (3) post any SB Product on any website, bulletin board, ftp server, newsgroup, or other similar mechanism or device; (4) remove from or alter these CIS Security Benchmarks Terms of Use on any SB Product; (5) remove or alter any proprietary notices on any SB Product; (6) use any SB Product or any component of an SB Product with any derivative works based directly on an SB Product or any component of an SB Product; (7) use any SB Product or any component of an SB Product with other products or applications that are directly and specifically dependent on such SB Product or any component for any part of their functionality; (8) represent or claim a particular level of compliance or consistency with any SB Product; or (9) facilitate or otherwise aid other individuals or entities in violating these CIS Security Benchmarks Terms of Use.
- **Your Responsibility to Evaluate Risks**. You acknowledge and agree that: (1) no network, system, device, hardware, software, or component can be made fully secure; (2) you have the sole responsibility to evaluate the risks and benefits of the SB Products to your particular circumstances and requirements; and (3) CIS is not assuming any of the liabilities associated with your use of any or all of the SB Products.
- **CIS Liability**. You acknowledge and agree that neither CIS nor any of its employees, officers, directors, agents or other service providers has or will have any liability to you whatsoever (whether based in contract, tort, strict liability or otherwise) for any direct, indirect, incidental, consequential, or special damages that arise out of or are connected in any way with your use of any SB Product.
- **Indemnification**. You agree to indemnify, defend, and hold CIS and all of CIS's employees, officers, directors, agents and other service providers harmless from and against any liabilities, costs and expenses incurred by any of them in connection with your violation of these CIS Security Benchmarks Terms of Use.
- **Jurisdiction**. You acknowledge and agree that: (1) these CIS Security Benchmarks Terms of Use will be governed by and construed in accordance with the laws of the State of Maryland; (2) any action at law or in equity arising out of or relating to these CIS Security Benchmarks Terms of Use shall be filed only in the courts located in the State of Maryland; and (3) you hereby consent and submit to the personal jurisdiction of such courts for the purposes of litigating any such action.
- **U.S. Export Control and Sanctions laws**. Regarding your use of the SB Products with any non-U.S. entity or country, you acknowledge that it is your responsibility to understand and abide by all U.S. sanctions and export control laws as set from time to time by the U.S. Bureau of Industry and Security (BIS) and the U.S. Office of Foreign Assets Control (OFAC).

SPECIAL RULES FOR CIS MEMBER ORGANIZATIONS: CIS reserves the right to create special rules for: (1) CIS Members; and (2) Non-Member organizations and individuals with which CIS has a written contractual relationship. CIS hereby grants to each CIS Member Organization in good standing the right to distribute the SB Products within such Member's own organization, whether by manual or electronic means. Each such Member Organization acknowledges and agrees that the foregoing grants in this paragraph are subject to the terms of such Member's membership arrangement with CIS and may, therefore, be modified or terminated by CIS at any time.

Table of Contents

Overview 7
Intended Audience 7
Consensus Guidance 7
Typographical Conventions 8
Scoring Information 8
Profile Definitions 9
Acknowledgements 10
Recommendations 11
1 Host Configuration 11
1.1 Create a separate partition for containers (Scored) 11
1.2 Use the updated Linux Kernel (Scored) 12
1.3 Do not use development tools in production (Not Scored) 13
1.4 Harden the container host (Not Scored) 14
1.5 Remove all non-essential services from the host (Not Scored) 15
1.6 Keep Docker up to date (Not Scored) 16
1.7 Only allow trusted users to control Docker daemon (Scored) 17
1.8 Audit docker daemon (Scored) 18
1.9 Audit Docker files and directories - /var/lib/docker (Scored) 20
1.10 Audit Docker files and directories - /etc/docker (Scored) 21
1.11 Audit Docker files and directories - docker-registry.service (Scored) 22
1.12 Audit Docker files and directories - docker.service (Scored) 23
1.13 Audit Docker files and directories - /var/run/docker.sock (Scored) 25
1.14 Audit Docker files and directories - /etc/sysconfig/docker (Scored) 26
1.15 Audit Docker files and directories - /etc/sysconfig/docker-network (Scored) 27

1.16 Audit Docker files and directories - /etc/sysconfig/docker-registry (Scored) .29
1.17 Audit Docker files and directories - /etc/sysconfig/docker-storage (Scored) ..30
1.18 Audit Docker files and directories - /etc/default/docker (Scored) 31
2 Docker daemon configuration 33
2.1 Do not use lxc execution driver (Scored) 33
2.2 Restrict network traffic between containers (Scored) 34
2.3 Set the logging level (Scored) 35
2.4 Allow Docker to make changes to iptables (Scored) 36
2.5 Do not use insecure registries (Scored) 37
2.6 Setup a local registry mirror (Scored) 38
2.7 Do not use the aufs storage driver (Scored) 39
2.8 Do not bind Docker to another IP/Port or a Unix socket (Scored) 40
2.9 Configure TLS authentication for Docker daemon (Scored) 42
2.10 Set default ulimit as appropriate (Not Scored) 43
3 Docker daemon configuration files 44
3.1 Verify that docker.service file ownership is set to root:root (Scored) 44
3.2 Verify that docker.service file permissions are set to 644 or more restrictive (Scored) 45
3.3 Verify that docker-registry.service file ownership is set to root:root (Scored) ...46
3.4 Verify that docker-registry.service file permissions are set to 644 or more restrictive (Scored) 47
3.5 Verify that docker.socket file ownership is set to root:root (Scored) 48
3.6 Verify that docker.socket file permissions are set to 644 or more restrictive (Scored) 49
3.7 Verify that Docker environment file ownership is set to root:root (Scored) 51
3.8 Verify that Docker environment file permissions are set to 644 or more restrictive (Scored) 52

3.9 Verify that docker-network environment file ownership is set to root:root (Scored) 53

3.10 Verify that docker-network environment file permissions are set to 644 or more restrictive (Scored) 54

3.11 Verify that docker-registry environment file ownership is set to root:root (Scored) 55

3.12 Verify that docker-registry environment file permissions are set to 644 or more restrictive (Scored) 56

3.13 Verify that docker-storage environment file ownership is set to root:root (Scored) 57

3.14 Verify that docker-storage environment file permissions are set to 644 or more restrictive (Scored) 58

3.15 Verify that /etc/docker directory ownership is set to root:root (Scored) 59

3.16 Verify that /etc/docker directory permissions are set to 755 or more restrictive (Scored) 60

3.17 Verify that registry certificate file ownership is set to root:root (Scored) 61

3.18 Verify that registry certificate file permissions are set to 444 or more restrictive (Scored) 62

3.19 Verify that TLS CA certificate file ownership is set to root:root (Scored) 63

3.20 Verify that TLS CA certificate file permissions are set to 444 or more restrictive (Scored) 64

3.21 Verify that Docker server certificate file ownership is set to root:root (Scored) 65

3.22 Verify that Docker server certificate file permissions are set to 444 or more restrictive (Scored) 66

3.23 Verify that Docker server certificate key file ownership is set to root:root (Scored) 67

3.24 Verify that Docker server certificate key file permissions are set to 400 (Scored) 68

3.25 Verify that Docker socket file ownership is set to root:docker (Scored) 69

3.26 Verify that Docker socket file permissions are set to 660 or more restrictive (Scored) 71
4 Container Images and Build File 72
4.1 Create a user for the container (Scored) 72
4.2 Use trusted base images for containers (Not Scored) 74
4.3 Do not install unnecessary packages in the container (Not Scored) 75
4.4 Rebuild the images to include security patches (Not Scored) 76
5 Container Runtime 78
5.1 Verify AppArmor Profile, if applicable (Scored) 78
5.2 Verify SELinux security options, if applicable (Scored) 79
5.3 Verify that containers are running only a single main process (Scored) 80
5.4 Restrict Linux Kernel Capabilities within containers (Scored) 81
5.5 Do not use privileged containers (Scored) 83
5.6 Do not mount sensitive host system directories on containers (Scored) 84
5.7 Do not run ssh within containers (Scored) 85
5.8 Do not map privileged ports within containers (Scored) 87
5.9 Open only needed ports on container (Scored) 88
5.10 Do not use host network mode on container (Scored) 89
5.11 Limit memory usage for container (Scored) 91
5.12 Set container CPU priority appropriately (Scored) 92
5.13 Mount container's root filesystem as read only (Scored) 94
5.14 Bind incoming container traffic to a specific host interface (Scored) 96
5.15 Set the 'on-failure' container restart policy to 5 (Scored) 97
5.16 Do not share the host's process namespace (Scored) 99
5.17 Do not share the host's IPC namespace (Scored) 100

5.18 Do not directly expose host devices to containers (Not Scored) 101
5.19 Override default ulimit at runtime only if needed (Not Scored) 103
6 Docker Security Operations 104
6.1 Perform regular security audits of your host system and containers (Not Scored) 104
6.2 Monitor Docker containers usage, performance and metering (Not Scored) 105
6.3 Endpoint protection platform (EPP) tools for containers (Not Scored) 106
6.4 Backup container data (Not Scored) 107
6.5 Use a centralized and remote log collection service (Not Scored) 109
6.6 Avoid image sprawl (Not Scored) 110
6.7 Avoid container sprawl (Not Scored) 112
Appendix: Summary Table 114
Appendix: Change History 118

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 1.6 or later technology.

Intended Audience

This document, CIS Docker 1.6 Benchmark, provides prescriptive guidance for establishing a secure configuration posture for Docker container version 1.6. This guide was tested against Docker 1.6.0 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 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, VMware, Inc.

Contributor
Adam Montville, Center for Internet Security
David Somers-Harris, Rakuten
Diogo Monica, Docker
Gregory Frascadore, VMware
Jerome Petazzoni, Docker
Kesten Broughton, Cognitive Scale
Paul Morgan, International Securities Exchange

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 Use the updated Linux Kernel (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Docker in daemon mode has specific kernel requirements. A 3.10 Linux kernel is the minimum requirement for Docker.

Rationale:

Kernels older than 3.10 lack some of the features required to run Docker containers. These older versions are known to have bugs which cause data loss and frequently panic under certain conditions. The latest minor version (3.x.y) of the 3.10 (or a newer maintained version) Linux kernel is thus recommended. Additionally, using the updated Linux kernels ensures that critical kernel bugs found earlier are fixed.

Audit:

Execute the below command to find out Linux kernel version:

```
uname -r
```

Ensure that the kernel version found is `3.10` or newer.

Remediation:

Check out the Docker kernel and OS requirements and suitably choose your kernel and OS.

Impact:

None.

Default Value:

Not Applicable.

References:

1. https://docs.docker.com/installation/binaries/#check-kernel-dependencies
2. https://docs.docker.com/installation/#installation-list

1.3 Do not use development tools in production (Not Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Development tools are not meant to be used in production.

Rationale:

Development tools usually provide alternate and convenient methods of using a technology. These tools are just meant to develop quick proof-of-concept or provide leads for up-selling of production ready software. There are various development tools that can be used with Docker such as boot2docker, Kitematic, VMware Fusion, Vagrant and others. Do not use these tools in production environment.

Audit:

Inspect the environment that is running Docker and verify that development tools are not used in production.

Remediation:

Do not use development tools in production.

Impact:

None.

Default Value:

Not Applicable.

References:

1. https://github.com/boot2docker/boot2docker
2. https://kitematic.com/

1.4 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. http://blog.dotcloud.com/kernel-secrets-from-the-paas-garage-part-44-g
5. https://grsecurity.net/
6. https://en.wikibooks.org/wiki/Grsecurity
7. https://pax.grsecurity.net/
8. http://en.wikipedia.org/wiki/PaX

1.5 Remove all non-essential services from the host (Not Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Ensure that the host running the docker daemon is running only the essential services.

Rationale:

It is a good practice to implement only one primary function per server to prevent functions that require different security levels from co-existing on the same server. Additionally, mixing various application environments on the same machine may hinder the granular administration of the respective applications.

Audit:

Inspect the docker host and ensure that it is exclusively used for running docker containers. Other services must not be found. Examples of other services include Web Server, database, or any function other than the container's main process. Some of the things you can do to inspect the system for other services are as below:

Check running processes:

```
ps -ef
```

Check socket information:

```
ss -nlp
```

or

```
netstat -nlp
```

Check inventory:

```
rpm -qa (or equivalent)
```

Remediation:

Move all other services within containers controlled by Docker or to other systems.

Impact:

None.

Default Value:

Not Applicable.

References:

1. http://blog.docker.com/2013/08/containers-docker-how-secure-are-they/

1.6 Keep Docker up to date (Not Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

The docker container solution is evolving to maturity and stability at a rapid pace. Like any other software, the vendor releases regular updates 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.

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.

Audit:

Execute the below command and verify that the Docker version is up to date.

```
docker version
```

Remediation:

Download and install the updated Docker software from official Docker repository.

Impact:

None.

Default Value:

Not Applicable

References:

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

1.7 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.

```
cat /etc/group | grep 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

1.8 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

1.9 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

1.10 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

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

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `/usr/lib/systemd/system/docker-registry.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. `/usr/lib/systemd/system/docker-registry.service` is one such file. It holds various parameters for Docker registries. It must be audited, if applicable.

Audit:

Verify that there is an audit rule corresponding to `/usr/lib/systemd/system/docker-registry.service` file.

For example, execute below command:

```
auditctl -l | grep /usr/lib/systemd/system/docker-registry.service
```

This should list a rule for `/usr/lib/systemd/system/docker-registry.service` file.

Remediation:

Add a rule for `/usr/lib/systemd/system/docker-registry.service` file.

For example,

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

```
-w /usr/lib/systemd/system/docker-registry.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 `/usr/lib/systemd/system/docker-registry.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

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

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `/usr/lib/systemd/system/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. `/usr/lib/systemd/system/docker.service` 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 `/usr/lib/systemd/system/docker.service` file.

For example, execute below command:

```
auditctl -l | grep /usr/lib/systemd/system/docker.service
```

This should list a rule for `/usr/lib/systemd/system/docker.service` file.

Remediation:

Add a rule for `/usr/lib/systemd/system/docker.service` file.

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

1.13 Audit Docker files and directories - /var/run/docker.sock (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `/var/run/docker.sock`.

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/run/docker.sock` is one such file. It is the default Docker Unix socket. The '`root`' and members of '`docker`' group interact with it for carrying out various Docker related activities. It must be audited.

Audit:

Verify that there is an audit rule corresponding to `/var/run/docker.sock` file.

For example, execute below command:

```
auditctl -l | grep /var/run/docker.sock
```

This should list a rule for `/var/run/docker.sock` file.

Remediation:

Add a rule for `/var/run/docker.sock` file.

For example,

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

```
-w /var/run/docker.sock -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

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

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `/etc/sysconfig/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/sysconfig/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/sysconfig/docker` file.

For example, execute below command:

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

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

Remediation:

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

For example,

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

```
-w /etc/sysconfig/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/sysconfig/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

1.15 Audit Docker files and directories - /etc/sysconfig/docker-network (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `/etc/sysconfig/docker-network`, 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/sysconfig/docker-network` is one such file. It holds various parameters for Docker daemon networking options. It must be audited, if applicable.

Audit:

Verify that there is an audit rule corresponding to `/etc/sysconfig/docker-network` file.

For example, execute below command:

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

This should list a rule for `/etc/sysconfig/docker-network` file.

Remediation:

Add a rule for `/etc/sysconfig/docker-network` file.

For example,

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

```
-w /etc/sysconfig/docker-network -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/sysconfig/docker-network` 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

1.16 Audit Docker files and directories - /etc/sysconfig/docker-registry (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `/etc/sysconfig/docker-registry`, 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/sysconfig/docker-registry` is one such file. It holds various parameters for Docker registry options. It must be audited, if applicable.

Audit:

Verify that there is an audit rule corresponding to `/etc/sysconfig/docker-registry` file.

For example, execute below command:

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

This should list a rule for `/etc/sysconfig/docker-registry` file.

Remediation:

Add a rule for `/etc/sysconfig/docker-registry` file.

For example,

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

```
-w /etc/sysconfig/docker-registry -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/sysconfig/docker-registry` 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

1.17 Audit Docker files and directories - /etc/sysconfig/docker-storage (Scored)

Profile Applicability:

- Level 1 - Linux Host OS

Description:

Audit `/etc/sysconfig/docker-storage`, 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/sysconfig/docker-storage` is one such file. It holds various parameters for Docker daemon storage options. It must be audited, if applicable.

Audit:

Verify that there is an audit rule corresponding to `/etc/sysconfig/docker-storage` file.

For example, execute below command:

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

This should list a rule for `/etc/sysconfig/docker-storage` file.

Remediation:

Add a rule for `/etc/sysconfig/docker-storage` file.

For example,

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

```
-w /etc/sysconfig/docker-storage -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/sysconfig/docker-storage` 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

1.18 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

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 Ubuntu. Also, note that Docker in daemon mode can be identified as having '`-d`' as the argument to `docker` service.

2.1 Do not use lxc execution driver (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

The default Docker execution driver is '`libcontainer`'. LXC as an execution driver is optional and just has legacy support.

Rationale:

There is still legacy support for the original LXC userspace tools via the '`lxc`' execution driver, however, this is not where the primary development of new functionality is taking place. Docker out of the box can now manipulate namespaces, control groups, capabilities, apparmor profiles, network interfaces and firewalling rules - all in a consistent and predictable way, and without depending on LXC or any other userland package. This drastically reduces the number of moving parts, and insulates Docker from the side-effects introduced across versions and distributions of LXC.

Audit:

```
$> ps -ef | grep docker | grep lxc
```

The above command should not return anything. This would ensure that the '`-e`' or '`--exec-driver`' parameter is either not present or not set to '`lxc`'.

Remediation:

Do not run the Docker daemon with '`lxc`' as execution driver.

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

```
$> docker -d --exec-driver=lxc
```

Impact:

None.

Default Value:

By default, Docker execution driver is '`libcontainer`'.

References:

1. http://www.infoq.com/news/2014/03/docker_0_9
2. http://docs.docker.com/reference/commandline/cli/#docker-exec-driver-option

2.2 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 docker
```

Ensure that the '`--icc`' parameter is set to '`false`'.

Remediation:

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

For Example,

```
$> docker -d --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

2.3 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 docker
```

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:

```
$> docker -d --log-level="info"
```

Impact:

None.

Default Value:

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

References:

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

2.4 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 docker
```

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:

```
$> docker -d --iptables=false
```

Impact:

None.

Default Value:

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

References:

1. http://docs.docker.com/articles/networking/#communication-between-containers

2.5 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:

Execute the below command to find out if any insecure registries are used:

```
$> ps -ef | grep docker
```

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:

```
$> docker -d --insecure-registry 10.1.0.0/16
```

Impact:

None.

Default Value:

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

References:

1. http://docs.docker.com/reference/commandline/cli/#insecure-registries

2.6 Setup a local registry mirror (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

The local registry mirror is serves the images from its own storage.

Rationale:

If you have multiple instances of Docker running in your environment, each time one of them requires an image, it will have to go out to the internet and fetch it from public or your private Docker registry. By running a local registry mirror, you can keep image fetch traffic on your local network. So, your Docker instances need not have to be internet facing and thus this drastically reduces the threat vector. Additionally, it allows you to manage and securely store your images within your own environment.

Audit:

Execute the below command to find out if a local registry is used:

```
$> ps -ef | grep docker
```

Ensure that the '`--registry-mirror`' parameter is present.

Remediation:

Configure a local registry mirror and then start the Docker daemon as below:

$> docker --registry-mirror=<registry path> -d

For example,

```
$> docker --registry-mirror=https://10.0.0.2:5000 -d
```

Impact:

The local registry mirror would need to be managed. It must have verified images that you use in your environment and those images must be kept updated time to time.

Default Value:

By default, there are no local registry mirrors setup on the host with Docker installation.

References:

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

2.7 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:

```
$> docker -s aufs -d
```

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. https://github.com/docker/docker/issues/6047
3. http://muehe.org/posts/switching-docker-from-aufs-to-devicemapper/
4. http://jpetazzo.github.io/assets/2015-03-05-deep-dive-into-docker-storage-drivers.html#1

2.8 Do not bind Docker to another IP/Port or a Unix socket (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. Do not bind Docker daemon to another IP/Port or a Unix socket.

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.

Audit:

```
$> ps -ef | grep docker
```

Ensure that the '`-H`' parameter is not present.

Remediation:

Do not bind the Docker daemon to any IP and Port or a non-default Unix socket.

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

```
$> docker -H tcp://10.1.2.3:2375 -H unix:///var/run/example.sock -d
```

Impact:

No one can have full access to Docker daemon except '`root`'. Alternatively, you should configure the TLS authentication for Docker and Docker Swarm APIs if you want to bind the Docker daemon to any other IP and Port.

Default Value:

By default, Docker daemon binds to a non-networked Unix socket.

References:

1. https://docs.docker.com/articles/basics/#bind-docker-to-another-hostport-or-a-unix-socket

2.9 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 docker
```

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. http://www.hnwatcher.com/r/1644394/Intro-to-Docker-Swarm-Part-2-Comfiguration-Modes-and-Requirements
3. http://www.blackfinsecurity.com/docker-swarm-with-tls-authentication/

2.10 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 docker
```

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,

```
$> docker -d --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. http://docs.docker.com/reference/commandline/cli/#default-ulimits

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:

If you are using Docker on a machine that uses systemd to manage services, then verify that the '`docker.service`' file ownership and group-ownership is 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:

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

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

The above command should not return anything.

Remediation:

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

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. 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/systemd/

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

Profile Applicability:

- Level 1 - Docker

Description:

If you are using Docker on a machine that uses systemd to manage services, then 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:

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

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

Remediation:

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

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

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-registry.service file ownership is set to root:root (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

If you are using Docker on a machine that uses systemd to manage services, then verify that the '`docker-registry.service`' file ownership and group-ownership is correctly set to '`root`'.

Rationale:

'`docker-registry.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:

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

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

The above command should not return anything.

Remediation:

```
#> chown root:root /usr/lib/systemd/system/docker-registry.service
```

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. 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/systemd/
2. https://github.com/docker/docker-registry

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

Profile Applicability:

- Level 1 - Docker

Description:

If you are using Docker on a machine that uses systemd to manage services, then verify that the '`docker-registry.service`' file permissions are correctly set to '`644`' or more restrictive.

Rationale:

'`docker-registry.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:

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

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

Remediation:

```
#> chmod 644 /usr/lib/systemd/system/docker-registry.service
```

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

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/
2. https://github.com/docker/docker-registry

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

Profile Applicability:

- Level 1 - Docker

Description:

If you are using Docker on a machine that uses systemd to manage services, then 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:

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

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

The above command should not return anything.

Remediation:

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

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. 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/

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

Profile Applicability:

- Level 1 - Docker

Description:

If you are using Docker on a machine that uses systemd to manage services, then 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:

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

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

Remediation:

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

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. 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/

3.7 Verify that Docker environment file ownership is set to root:root (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Docker daemon leverages Docker environment file for setting Docker daemon run time environment. If you are using Docker on a machine that uses systemd to manage services, then the file is `/etc/sysconfig/docker`. On other systems, the environment file is `/etc/default/docker`. Verify that the environment file ownership and group-ownership is correctly set to '`root`'.

Rationale:

Docker environment file contains sensitive parameters that may alter the behavior of Docker daemon during run time. 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 environment file is owned and group-owned by '`root`':

```
stat -c %U:%G <Environment file name> | grep -v root:root
```

For example,

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

The above command should not return anything.

Remediation:

```
#> chown root:root <Environment file name>
```

For example,

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

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

Impact:

None.

Default Value:

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

References:

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

3.8 Verify that Docker environment file permissions are set to 644 or more restrictive (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Docker daemon leverages Docker environment file for setting Docker daemon run time environment. If you are using Docker on a machine that uses systemd to manage services, then the file is `/etc/sysconfig/docker`. On other systems, the environment file is `/etc/default/docker`. Verify that the environment file permissions are correctly set to '`644`' or more restrictive.

Rationale:

Docker environment file contains sensitive parameters that may alter the behavior of Docker daemon during run time. Hence, it should be only writable by '`root`' to maintain the integrity of the file.

Audit:

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

```
stat -c %a <Environment file name>
```

For example,

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

Remediation:

```
#> chmod 644 <Environment file name>
```

For example,

```
#> chmod 644 /etc/sysconfig/docker
```

This would set the file permissions for the environment file to '644'.

Impact:

None.

Default Value:

By default, the file permissions for this file is correctly set to '644'.

References:

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

3.9 Verify that docker-network environment file ownership is set to root:root (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

If you are using Docker on a machine that uses systemd to manage services, then verify that the 'docker-network' file ownership and group-ownership is correctly set to 'root'.

Rationale:

'docker-network' 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/sysconfig/docker-network | grep -v root:root
```

The above command should not return anything.

Remediation:

```
#> chown root:root /etc/sysconfig/docker-network
```

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. 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/systemd/

3.10 Verify that docker-network environment file permissions are set to 644 or more restrictive (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

If you are using Docker on a machine that uses systemd to manage services, then verify that the '`docker-network`' file permissions are correctly set to '`644`' or more restrictive.

Rationale:

'`docker-network`' 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:

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

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

Remediation:

```
#> chmod 644 /etc/sysconfig/docker-network
```

This would set the file permissions to '644'.

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/

3.11 Verify that docker-registry environment file ownership is set to root:root (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

If you are using Docker on a machine that uses systemd to manage services, then verify that the 'docker-registry' file ownership and group-ownership is correctly set to 'root'.

Rationale:

'docker-registry' 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/sysconfig/docker-registry | grep -v root:root
```

The above command should not return anything.

Remediation:

```
#> chown root:root /etc/sysconfig/docker-registry
```

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. 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/systemd/

3.12 Verify that docker-registry environment file permissions are set to 644 or more restrictive (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

If you are using Docker on a machine that uses systemd to manage services, then verify that the '`docker-registry`' file permissions are correctly set to '`644`' or more restrictive.

Rationale:

'`docker-registry`' 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:

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

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

Remediation:

```
#> chmod 644 /etc/sysconfig/docker-registry
```

This would set the file permissions to '644'.

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/

3.13 Verify that docker-storage environment file ownership is set to root:root (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

If you are using Docker on a machine that uses systemd to manage services, then verify that the 'docker-storage' file ownership and group-ownership is correctly set to 'root'.

Rationale:

'docker-storage' 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/sysconfig/docker-storage | grep -v root:root
```

The above command should not return anything.

Remediation:

```
#> chown root:root /etc/sysconfig/docker-storage
```

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. 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/systemd/

3.14 Verify that docker-storage environment file permissions are set to 644 or more restrictive (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

If you are using Docker on a machine that uses systemd to manage services, then verify that the '`docker-storage`' file permissions are correctly set to '`644`' or more restrictive.

Rationale:

'`docker-storage`' 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:

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

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

Remediation:

```
#> chmod 644 /etc/sysconfig/docker-storage
```

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

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/

3.15 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/

3.16 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/

3.17 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

3.18 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

3.19 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/

3.20 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 be 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/

3.21 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/

3.22 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 be 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/

3.23 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/

3.24 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/

3.25 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

3.26 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

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. Also, run the container with non-root user.

Rationale:

Currently, mapping the container's root user to a non-root user on the host is not supported by Docker. The support for user namespace would be provided in future releases (probably in 1.6). This creates a serious user isolation issue. It is thus highly recommended to ensure that there is a non-root user created for the container and the container is run using that user.

Audit:

```
docker ps -q | 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
```

When you run the container, use the '`-u`' flag to specify that you would want to run the container as a specific user and not `root`. This can be done by executing below command:

$> docker run -u <Username or ID> <Run args> <Container Image Name or ID> <Command>

For example,

```
$> docker run -u 1000 -i -t centos /bin/bash
```

This would ensure that the above container is run with user ID 1000 and not root.

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. But, the Docker container image signing and verification feature is not yet ready. Hence, the Docker engine does not verify the provenance of the container images by itself. You should thus exercise a great deal 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:

Only download the container images from a source you trust over a secure channel. Additionally, use features such as pull-by-digest to get specific images from the registry.

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

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 -q
```

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 Rebuild the images to include security patches (Not Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Instead of patching your containers and images, rebuild the images from scratch and instantiate new containers from it.

Rationale:

Security patches are updates to products to resolve known issues. These 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.

On conventional systems, rebuilding the system is a risky operation, because it is rarely done, and many components can diverge over time between two rebuilds. However, when using state-of-the-art configuration management or a build system like the one provided by Docker with Dockerfiles, rebuilds are easy to do, and fast. This allows (and promotes) frequent rebuilds, and ensures that at any given time, it is safe (and reliable) to do such a full rebuilds. As such, in case of security vulnerability affecting a package or library, rebuilding with the latest available software is the best practice and should be followed.

Audit:

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

```
docker ps -q
```

Step 2: For each container instance, execute the below or equivalent command to find the list of packages installed within container. Ensure that the security updates for various affected packages are installed.

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

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 version).
Step 2: Force a rebuild of each image with '`docker build --no-cache`'.
Step 3: Restart all containers with the updated images.

Impact:

Rebuilding the images has to be done after upstream packages are available, otherwise re-pulling and rebuilding will do no good. When the affected packages are in the base image, it is necessary to pull it (and therefore rebuild). When the affected packages are in the downloaded packages, it is not necessary to pull the image; but nonetheless, in doubt, it is recommended to always follow this strict procedure and rebuild the entire image.

Note: If updated packages are not available and it is critical to install a security patch, live patching could be used.

Default Value:

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

References:

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

5 Container Runtime

The ways in which a container is started governs a lot 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 Verify AppArmor Profile, if applicable (Scored)

Profile Applicability:

- Level 2 - 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 should create a AppArmor profile for your containers. This would enforce security policies on the containers as defined in the profile.

Audit:

```
docker ps -q | 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 Docker AppArmor profile. For example,

```
docker run -i -t --security-opt="apparmor:PROFILENAME" centos /bin/bash
```

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, no AppArmor profiles 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://wiki.apparmor.net/index.php/Main_Page

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 -q | 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 -d --selinux-enabled
```

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

```
docker run -i -t --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 Verify that containers are running only a single main process (Scored)

Profile Applicability:

- Level 1 - Docker

Description:

In almost all cases, you should only run a single main process (that main process could spawn children, which is ok) in a single container. Decoupling applications into multiple containers makes it much easier to scale horizontally and reuse containers. If that service depends on another service, make use of container linking.

Rationale:

By design, Docker watches one single process within the container. So, installing and running multiple applications within a single container *breaks* the basic design of 'one container one process'.

If you need multiple processes, you need to add one at the top-level to take care of the others. You also need to add a process manager; for instance `Monit` or `Supervisor`. In other words, you're turning a lean and simple container into something much more complicated. If your application stops (if it exits cleanly or if it crashes), instead of getting that information through Docker, you will have to get it from your process manager.

Audit:

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

```
docker ps -q
```

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

```
docker exec $INSTANCE_ID ps -el
```

Ensure that there is only one process running i.e. the process the container is intended to run. There should not be a process for process manager such as `Monit, Supervisor` or any other.

Remediation:

Do not run multiple applications within a single container. Use container linking instead to run multiple applications in multiple containers in tandem.

Impact:

None.

Default Value:

By default, only one process per container is allowed.

References:

1. https://docs.docker.com/articles/dockerfile_best-practices
2. https://docs.docker.com/userguide/dockerlinks
3. https://docs.docker.com/articles/using_supervisord

5.4 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 -q | 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 --cap-add={"NET_ADMIN","SYS_ADMIN"} -i -t 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 --cap-drop={"SETUID","SETGID"} -i -t 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

5.5 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 -q | 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 --privileged -i -t 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

5.6 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 -q | xargs docker inspect --format '{{ .Id }}: Volumes={{ .Volumes }}
VolumesRW={{ .VolumesRW }}'
```

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

5.7 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 -q
```

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 -i -t $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/

5.8 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:

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

```
docker ps -q
```

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

```
docker port $INSTANCE_ID
```

The above command would list all the container to host port mappings. 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 are 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

5.9 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:

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

```
docker ps -q
```

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

```
docker port $INSTANCE_ID
```

The above command would list all the container to host port mappings. 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) flag when starting the container. Use the '`-p`' (lowercase) flag to explicitly define the ports that you need for a particular container instance.

For example,

```
$> docker run -i -t -p 5000 -p 5001 -p 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`' flag.

References:

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

5.10 Do not use host network mode on container (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 -q | xargs docker inspect --format '{{ .Id }}: NetworkMode={{
.HostConfig.NetworkMode }}'
```

If the above command returns '`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

5.11 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 -q | xargs docker inspect --format '{{ .Id }}: Memory={{ .Config.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 '`-m`' argument. You should start the container as below:

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

For example,

```
$> docker run -i -t -m 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/

5.12 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 -q | xargs docker inspect --format '{{ .Id }}: CpuShares={{
.Config.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 '`-c`' argument. You may start the container as below:

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

For example,

```
$> docker run -i -t -c 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`' 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 maths 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/

5.13 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 -q | 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 -i -t --read-only -v /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

5.14 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:

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

```
docker ps -q
```

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

```
docker port $INSTANCE_ID
```

The above command would list all the container to host interface and port mappings. 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`.

```
80/tcp -> 0.0.0.0:49153
```

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.

```
80/tcp -> 10.2.3.4:49153
```

Remediation:

Bind the container port to a specific host interface on the desired host port.

For example,

```
$> docker run -d -p 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

5.15 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 -q | 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 -d --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

5.16 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 -q | 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 -i -t --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

5.17 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 -q | 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 -i -t --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 -i -t --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

5.18 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 -q | 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 -i -t --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 -i -t --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

5.19 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 -q | 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 -i -t 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

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:

Verify whether a container or a software is used for tracking container usage, reporting performance and metering.

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 Endpoint protection platform (EPP) tools for containers (Not Scored)

Profile Applicability:

- Level 2 - Docker

Description:

There is no container-aware endpoint protection platform (EPP) solution as of now. You must rely on compensating controls to achieve the same.

Rationale:

Traditional EPP and encryption vendors have not yet recognized containers as an area that they need to pursue and secure in the future. Hence, there are no suitable products at this time. Thus, you must rely on compensating controls.

Audit:

Verify whether compensating controls, in place of traditional EPP, exist. Some of the compensating controls might be as below:

- Application white-listing (as supported by AppArmor)
- Mandatory access controls (as supported by SELinux) or
- Make containers self-assessing entities using the DevOps tool chain

Remediation:

AppArmor, SELinux and DevOps product configurations for containers are beyond the scope of this benchmark. You should seek guidance on specific configuration needed for containers from their respective sources.

Impact:

None.

Default Value:

By default, no endpoint protection is provided to containers.

References:

1. https://www.gartner.com/doc/2956826/security-properties-containers-managed-docker

6.4 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

6.5 Use a centralized and remote log collection service (Not Scored)

Profile Applicability:

- Level 1 - Docker

Description:

Each container maintains its logs under `/var/lib/docker/containers/$INSTANCE_ID/$INSTANCE_ID-json.log`. But, maintaining logs at a centralized place is preferable.

Rationale:

Storing log data on a remote host or a centralized place protects log integrity from local attacks. If an attacker gains access on the local system, he could tamper with or remove log data that is stored on the local system. Also, the '`docker logs`' paradigm is not yet fully developed. There are quite a few difficulties in managing the container logs namely

- No logrotate for container logs
- Transient behavior of docker logs
- Difficulty in accessing application specific log files
- All stdout and stderr are logged

Hence, a centralized and remote log collection service should be utilized to keep logs for all the containers.

Audit:

First, verify that the centralized and remote log collection service is configured. Then verify that all the containers are logging at this centralized place.

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

```
docker ps -q
```

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

```
docker inspect --format='{{.Volumes}}' $INSTANCE_ID
```

Ensure that a centralized log volume is mounted on the containers.

Remediation:

Configure a centralized and remote log collection service. Some of the examples to do this are in references. Once the log collection service is active, configure all the containers to send their logs to this service.

Impact:

None.

Default Value:

By default, each container logs separately.

References:

1. https://docs.docker.com/reference/commandline/cli/#logs
2. http://jpetazzo.github.io/2014/08/24/syslog-docker/
3. http://stackengine.com/docker-logs-aggregating-ease/

6.6 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 ps -q | xargs docker inspect --format '{{ .Id }}: Image={{ .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 ps -q | xargs docker inspect --format '{{ .Id }}: Image={{ .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

6.7 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 the below command to find the total number of containers you have on the host:

```
docker info | grep "Containers"
```

Now, execute the below command to find the total number of containers that are actually running on the host.

```
docker ps -q | wc -l
```

If the difference between the number of containers that are present 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 containers that are not needed using the below command:

```
$> docker rm $INSTANCE_ID
```

For example,

```
$> docker rm e3a7a1a97c58
```

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	Use the updated Linux Kernel (Scored)	☐	☐
1.3	Do not use development tools in production (Not Scored)	☐	☐
1.4	Harden the container host (Not Scored)	☐	☐
1.5	Remove all non-essential services from the host (Not Scored)	☐	☐
1.6	Keep Docker up to date (Not Scored)	☐	☐
1.7	Only allow trusted users to control Docker daemon (Scored)	☐	☐
1.8	Audit docker daemon (Scored)	☐	☐
1.9	Audit Docker files and directories - /var/lib/docker (Scored)	☐	☐
1.10	Audit Docker files and directories - /etc/docker (Scored)	☐	☐
1.11	Audit Docker files and directories - docker-registry.service (Scored)	☐	☐
1.12	Audit Docker files and directories - docker.service (Scored)	☐	☐
1.13	Audit Docker files and directories - /var/run/docker.sock (Scored)	☐	☐
1.14	Audit Docker files and directories - /etc/sysconfig/docker (Scored)	☐	☐
1.15	Audit Docker files and directories - /etc/sysconfig/docker-network (Scored)	☐	☐
1.16	Audit Docker files and directories - /etc/sysconfig/docker-registry (Scored)	☐	☐
1.17	Audit Docker files and directories - /etc/sysconfig/docker-storage (Scored)	☐	☐
1.18	Audit Docker files and directories - /etc/default/docker (Scored)	☐	☐
2	**Docker daemon configuration**		
2.1	Do not use lxc execution driver (Scored)	☐	☐
2.2	Restrict network traffic between containers (Scored)	☐	☐
2.3	Set the logging level (Scored)	☐	☐
2.4	Allow Docker to make changes to iptables (Scored)	☐	☐
2.5	Do not use insecure registries (Scored)	☐	☐
2.6	Setup a local registry mirror (Scored)	☐	☐
2.7	Do not use the aufs storage driver (Scored)	☐	☐
2.8	Do not bind Docker to another IP/Port or a Unix socket (Scored)	☐	☐
2.9	Configure TLS authentication for Docker daemon (Scored)	☐	☐

2.10	Set default ulimit as appropriate (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-registry.service file ownership is set to root:root (Scored)	☐	☐
3.4	Verify that docker-registry.service file permissions are set to 644 or more restrictive (Scored)	☐	☐
3.5	Verify that docker.socket file ownership is set to root:root (Scored)	☐	☐
3.6	Verify that docker.socket file permissions are set to 644 or more restrictive (Scored)	☐	☐
3.7	Verify that Docker environment file ownership is set to root:root (Scored)	☐	☐
3.8	Verify that Docker environment file permissions are set to 644 or more restrictive (Scored)	☐	☐
3.9	Verify that docker-network environment file ownership is set to root:root (Scored)	☐	☐
3.10	Verify that docker-network environment file permissions are set to 644 or more restrictive (Scored)	☐	☐
3.11	Verify that docker-registry environment file ownership is set to root:root (Scored)	☐	☐
3.12	Verify that docker-registry environment file permissions are set to 644 or more restrictive (Scored)	☐	☐
3.13	Verify that docker-storage environment file ownership is set to root:root (Scored)	☐	☐
3.14	Verify that docker-storage environment file permissions are set to 644 or more restrictive (Scored)	☐	☐
3.15	Verify that /etc/docker directory ownership is set to root:root (Scored)	☐	☐
3.16	Verify that /etc/docker directory permissions are set to 755 or more restrictive (Scored)	☐	☐
3.17	Verify that registry certificate file ownership is set to root:root (Scored)	☐	☐
3.18	Verify that registry certificate file permissions are set to 444 or more restrictive (Scored)	☐	☐
3.19	Verify that TLS CA certificate file ownership is set to root:root (Scored)	☐	☐
3.20	Verify that TLS CA certificate file permissions are set to 444 or more restrictive (Scored)	☐	☐
3.21	Verify that Docker server certificate file ownership is set to root:root (Scored)	☐	☐

3.22	Verify that Docker server certificate file permissions are set to 444 or more restrictive (Scored)	☐	☐
3.23	Verify that Docker server certificate key file ownership is set to root:root (Scored)	☐	☐
3.24	Verify that Docker server certificate key file permissions are set to 400 (Scored)	☐	☐
3.25	Verify that Docker socket file ownership is set to root:docker (Scored)	☐	☐
3.26	Verify that Docker socket file permissions are set to 660 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	Rebuild the images to include security patches (Not Scored)	☐	☐
5	**Container Runtime**		
5.1	Verify AppArmor Profile, if applicable (Scored)	☐	☐
5.2	Verify SELinux security options, if applicable (Scored)	☐	☐
5.3	Verify that containers are running only a single main process (Scored)	☐	☐
5.4	Restrict Linux Kernel Capabilities within containers (Scored)	☐	☐
5.5	Do not use privileged containers (Scored)	☐	☐
5.6	Do not mount sensitive host system directories on containers (Scored)	☐	☐
5.7	Do not run ssh within containers (Scored)	☐	☐
5.8	Do not map privileged ports within containers (Scored)	☐	☐
5.9	Open only needed ports on container (Scored)	☐	☐
5.10	Do not use host network mode on container (Scored)	☐	☐
5.11	Limit memory usage for container (Scored)	☐	☐
5.12	Set container CPU priority appropriately (Scored)	☐	☐
5.13	Mount container's root filesystem as read only (Scored)	☐	☐
5.14	Bind incoming container traffic to a specific host interface (Scored)	☐	☐
5.15	Set the 'on-failure' container restart policy to 5 (Scored)	☐	☐
5.16	Do not share the host's process namespace (Scored)	☐	☐
5.17	Do not share the host's IPC namespace (Scored)	☐	☐
5.18	Do not directly expose host devices to containers (Not Scored)	☐	☐
5.19	Override default ulimit at runtime only if needed (Not 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	Endpoint protection platform (EPP) tools for containers (Not Scored)	☐	☐
6.4	Backup container data (Not Scored)	☐	☐
6.5	Use a centralized and remote log collection service (Not Scored)	☐	☐
6.6	Avoid image sprawl (Not Scored)	☐	☐
6.7	Avoid container sprawl (Not Scored)	☐	☐

Appendix: Change History

Date	Version	Changes for this version
04-22-2015	1.0.0	Initial Release