Global Information Assurance Certification Paper

A Checklist for Audit of Docker Containers

GIAC GSNA Gold Certification

Author: Alyssa Robinson, lyssanr@yahoo.com
Advisor: Robert Vandenbrink
Accepted: 18 November, 2016

Abstract

Docker and other container technologies are increasingly popular methods for deploying applications in DevOps environments, due to advantages in portability, efficiency in resource sharing and speed of deployment. The very properties that make Docker containers useful, however, can pose challenges for audit, and the security capabilities and best practices are changing rapidly. As adoption of this technology grows, it is, therefore, necessary to create a standardized checklist for audit of Dockerized environments based on the latest tools and recommendations.

1. Why Docker? Why now?

After World War II, the original containerization technology provided a standard framework for shipping freight via railway, container ship, or truck. Freight in containers could be stored and moved between these transportation mechanisms, improving shipping speed while reducing costs (World Shipping Council, 2016). Similarly, Docker containers improve the speed of application deployment, (Docker, 2016) hiding the details of the OS, the network and other host-specific resources from developers (Wang, 2016) and providing the ability to ship an application seamlessly between environments (Wang, 2016). Containerization allows developers to package an application and all of its dependencies together in a standardized format, without the need to re-compile or to find and install the correct packages (Mouat, 2016b). Once a Dockerfile is built, it can be run across multiple Linux kernel versions, on hosted cloud platforms (Kearns, 2016), and now even on MacOS and Windows systems (Chanezon, 2016). This flexibility makes it easy to create a development environment that mimics production (Mouat, 2016b) on a smaller scale and easy to test an application once, rather than multiple times across different environments (Wang, 2016).

While Docker is gaining rapidly in popularity, with many companies experimenting with the technology, most haven't yet made the leap to using Docker for production deployments (Mouat, 2015a). A 2016 Cloud Foundry research report states that while only a small percentage of respondents were running containers in production now, many planned to deploy in production within the next year (Cloud Foundry, 2016). Cloud monitoring company Datadog has seen a 30% growth in the number of customers adopting Docker in just the last year, with an even higher rate of adoption for its larger customers (Datadog, 2016). Technologies for deploying and managing containers have multiplied and matured rapidly over the last few years (Mouat, 2016b) and multiple studies predict a high rate of continued growth over the next few years (Weins, 2016). As containers become mainstream, auditors will need to understand the best practices for securing and auditing this technology. Existing auditing tools are immature, but are evolving quickly in response to Docker growth (de la Fuente, 2015).

Alyssa Robinson, lyssanr@yahoo.com

2. What is Docker?

Docker provides a platform and accompanying tools for building, distributing and deploying applications using containers (Docker, 2016). Container technologies like Docker provide partitioned environments for running applications; whereas VMWare and other hardware virtualization technologies trick an OS into believing it is running on dedicated hardware, containers make an application believe it is running in a dedicated operating system (Haff, 2016). Several applications can thus share one kernel, reducing both resource and management overhead, while remaining isolated from one another (Docker, 2016).

To run a Docker deployment, a Docker daemon that manages the application containers is run on a physical or virtual host (Wang, 2016). The containers themselves are launched from an image built using a layered template file, referred to as a Dockerfile (Docker 2016). The "base" layer is the image on top of which the application container is built: at a minimum, an OS user-space snapshot, but the base could also be an application image (Docker, 2016). Text instructions are then layered on top of the base when the image is built, adding commands, packages, users, files and filesystems and specifying the processes to run when a container is launched (Mouat, 2015a). While the Docker image itself is immutable, running containers have a writable layer and can write out to volumes mounted from the host operating system (Wang, 2016).

Alyssa Robinson, lyssanr@yahoo.com

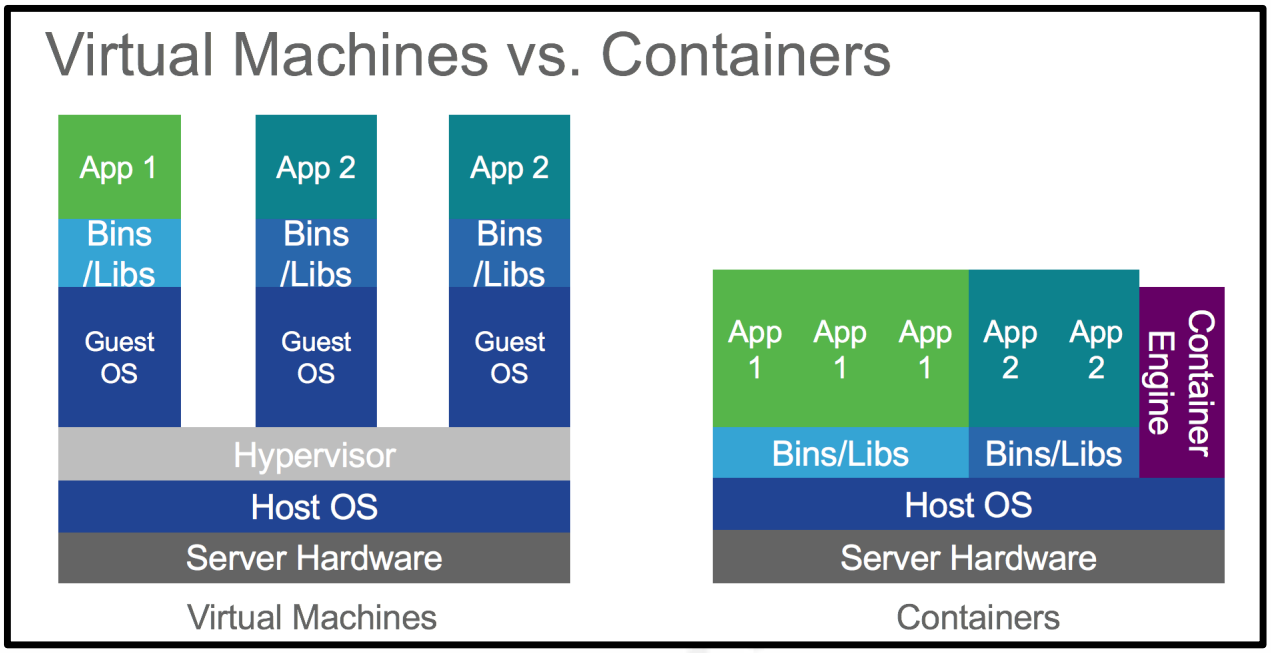


Figure 1 A comparison of Virtual Machine and Container architecture (Docker, 2016)

Once images are built, they can be distributed using Registry services, including the official Docker registry, known as the Docker Hub (Mouat, 2015b). While the Docker Hub repository provides access to hundreds of thousands of images for all sorts of applications and environments, running an in-house registry provides greater control over which images are used and who can see the images created internally (Docker, 2016). In-house registries can also provide improved performance over Internet download of images, as well as the ability to configure additional security features, such as image signing (Hausenblas, 2015). Images can be automatically uploaded to the registry, tagged for searching, and downloaded via Docker commands or additional orchestration (Mouat, 2015b).

3. Docker Security Issues

Red Hat's Dan Walsh quoted a coworker in a 2014 article on Docker security as saying, "Docker is about running random code downloaded from the Internet and running it as root" (Walsh, 2014). As the number of containers running on a system grows, so does the likelihood that one of the contained applications has a flaw that could lead to a breach, allowing container breakout (Mouat, 2015b). While Docker provides some level of resource and process isolation between containers, the fact remains that a process running as root within a container is running

as root on the host system, in direct communication with kernel subsystems and devices (Walsh, 2014). Tools for namespacing processes, scanning for vulnerabilities and identifying the provenance of containers exist, but are still maturing (Linthicum, 2015) and aren't standardized, adding to the challenges of managing containers at scale in a production environment.

4. Docker Audit Challenges

Auditing Docker deployments can be challenging based both on the Docker architecture itself and the microservices/continuous deployment models with which Docker aligns so well. According to application monitoring software provider New Relic, 46% of the containers they monitor last for less than one hour and 11% are deployed for under a minute (McGuire, 2015). These short lifetimes and the density of containers deployed make the task of monitoring significantly different than monitoring VM or hardware deployments (Datadog, 2016) and can lead to an asset management nightmare trying to track exactly what was deployed where when. Effective container audit thus looks at containers by role, rather than as individual entities, assuming deployment processes support this model (Datadog, 2016).

Additionally, security vulnerabilities may lie either within the container or in the host OS, meaning both must be tracked, patched, and auditable. Each of these environments generally requires different audit tools and processes (Cramer, 2016). With an estimated 30% of images in the Docker Hub showing vulnerabilities at any one time (Petazzoni, 2015), it's important to know what's inside a deployed version of a container at any given time. This isn't particularly easy for images that haven't been built in-house; even for those that have, software versions installed in a container built from the same Dockerfile can be different based just on the time it was built (Boettiger, 2014).

The microservice architecture and Continuous Delivery practices that generally go hand-in-hand with Docker deployments can have some significant advantages for auditors and for the companies that run them. Continuous Delivery depends on small changes, deployed frequently using an automated test infrastructure to reduce risk (Humble, 2010). If the security team can fit its checks into this process, the end result is a more secure product (Bellis, 2015). Automating auditing of security-related settings as part of this process allows the auditor to audit the process

itself, rather than the settings on each system or container; the process is where the real problems generally lie (Hoelzer, 2015). On the other side, using containers can free developers (Flower, 2015) from the normal security and operational processes that provide oversight or maintenance of what gets deployed to the production network.

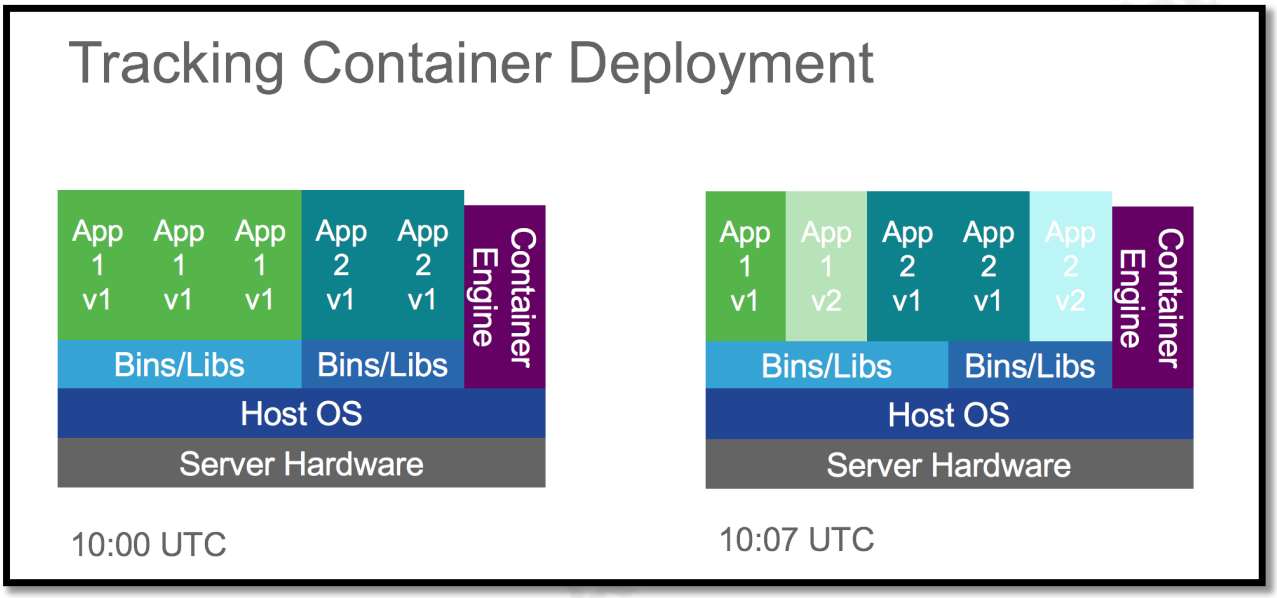


Figure 2 Container images and layout can change rapidly

5. Best Practices and Audit Checklist

5.1 Ensure good host security

In container deployments, as in the non-containerized world, good security relies upon multiple layers; a secure Docker implementation relies on the security of the host as well as the container implementations (Mouat, 2015b). The Docker host OS must be patched regularly and should follow best practices for securing the host OS (Gürkök, 2016). Kernel hardening enhancements like Grsecurity and Pax provide additional protections against container and host exploit (Boelen, 2015), as do properly configured AppArmor or SElinux, for which Docker templates are available (Docker, 2016). When multiple containers are deployed on a host, they should be segregated according to the sensitivity level of the hosted data, keeping a container breakout in a less secure service from leading to breach of more sensitive data (Mouat, 2015a). Docker is relatively immature and undergoing rapid development; it should be updated regularly to obtain security fixes and deprecated and unsupported features should be avoided (Mouat, 2015b).

Alyssa Robinson, lyssanr@yahoo.com

Checklist:

- ☐ Patch hosts
- ☐ Install only needed components
- ☐ Hardened kernel (Grsecurity, Pax)
- ☐ Apparmor/SElinux
- ☐ Segregate containers by data sensitivity
- ☐ Update Docker software regularly

5.2 Check Image Provenance

Using Docker images downloaded from Docker Hub or another registry provides multiple benefits in building an application, including increased speed and --in the case of "official" images-- the expertise of the image builders and the collective wisdom of previous users (Mouat, 2015b). Since running a poisoned image could compromise your infrastructure, understanding the provenance of images is incredibly important (Mouat, 2015a). Most of the open source licenses used by the images available on Docker Hub protect their creators from liability (Open Source Initiative), leaving providers that make use of these images liable for security problems (Koohgoli, 2014). Either when downloading from Docker Hub or building images, the Docker Content Trust feature ensures that images signatures and hashes are checked before use and requires that users sign their own images as well (Mouat, 2016).

Verifying container integrity is particularly important when images are transferred over an untrusted network: (Docker, 2016) image signing provides protection against any container tampering that could happen in transport (Gürkök, 2016). Both image signing and checking image signatures require a Docker Notary server. Both the root keys used to create signed repositories and repository keys used to sign images within those repositories need to be protected from unauthorized access and securely backed up. Root keys should always be stored offline; repository keys need strong passphrases and can be rotated or expired if needed (Docker, 2016).

When deploying images from a registry, the unique "digest" hash should also be used to ensure that the image pulled is the one that's been tested and has not been changed or

corrupted in transit (Mouat, 2015b). While Content Trust gives peace of mind that an image was created by a reputable provider, setting the explicit content hash ensures that the image in use is the specific image requested, not an earlier or later version (Docker, 2016).

Checklist:

- ☐ Enable DOCKER_CONTENT_TRUST environment variable, so that only signed images can be pulled.
- ☐ Store root key offline
- ☐ Backup signing keys; rotate & expire old keys
- ☐ Check image digest at deployment

5.3 Monitor Containers

Logging and monitoring are essential for identifying and investigating security incidents and providing audit trails (OWASP, 2016). Centralized logging becomes even more important in a containerized environment, since a given container may no longer be available when it's time to track down a breach or other issue (Mouat, 2015b). In addition to the concerns regarding ephemeral containers, Docker environments require logging at multiple layers -- the host, the Docker infrastructure, and the containers themselves -- to get the full view of any activity (Newman, 2015). By default, all Docker containers log to STDOUT/STDERR, with the logs made available via the 'Docker logs' command, but several strategies can be used to transfer logs to a permanent storage location. The "`–logdriver`" argument can make use of a logging driver such as Syslog or Fluentd, allowing for movement to a centralized log location via another process on the container or host (Mouat, 2015b). Alternately, a dedicated logging container (e.g. logspout) can also be used to retrieve and aggregate logs from multiple containers (Newman, 2015).

Checklist:

- ☐ Capture host logs
- ☐ Capture logs from Docker infrastructure
- ☐ Capture container logs
- ☐ Ensure adequate log information at the containers
    - Ensure –log-level is set to INFO (default) (CIS, 2016)

5.4 Do Not Run Container Processes as Root

Namespaces isolate processes running in one container from processes running in another container or on the host system (Docker, 2016). By default, the user namespace for containers is the same as that of the host. More specifically, the root user inside of a container is the root user of the host system and the compromise of a containerized process running as root has the potential to compromise the Docker host (Estest, 2015). Apps in containers should therefore not be run as root; instead, a non-privileged user should be created and processed run with a USER statement inside Dockerfiles (Mouat, 2015b). Having a unique user for each container allows an administrator to assign the minimum set of Linux capabilities required for a particular process (Docker, 2016) and provides per-user accounting (Estes, 2015).

Generally, containers do not need root privileges; root privileges are not even required for many of the services that required root in the non-containerized world. In a Docker environment, infrastructure services like network and hardware management, cron, and SSH are all provided either by Docker itself or the Docker host (Docker, 2016). Similarly, root privileges aren't required for container processes to bind a port in the 0-1024 range, as on the host and can instead be granted to the container via the NET_BIND_SERVICE capability (Docker, 2016).

Docker versions after 1.10 allow for User Namespace remap, allowing the container to have UIDs/GIDs that map into a different UID/GID range on the host using the Linux subordinate range file functionality (Estes, 2015). This capability includes the re-map of the root user inside a container to a non-UID 0 user outside of the container (Docker, 2016), but this approach does have limitations (Mouat, 2015b). One difficulty with the Docker User Namespace feature is that the UID/GID mapping happens at the Docker daemon level, rather than at the per-container level; this is due to limitations in the ability to shift the UIDs for filesystem mount (Estes, 2015).

Access to the Docker CLI is an all or nothing prospect, with members of the Docker group able to start and stop containers and change their configuration (Kuusik, 2015). As

such, access to the Docker daemon itself should also be strictly limited: any user with access to the Docker group essentially has privileged access not only to the containers, but also to the host system (Nieto, 2015). Setuid and Setgid binaries should also be removed from images, lessening the chance of privilege escalation if there are exploitable vulnerabilities in these utilities (Mouat, 2015b).

Checklist:

- ☐ Container processes run as non-privileged USER
- ☐ If process must run as root, use Docker User Namespace feature to re-map to non-privileged user on host
- ☐ Limit access to run Docker
- ☐ Remove setUID/setGID binaries

5.5 Do Not Store Secrets in Containers

Application containers may need database credentials, SSL certificates and private keys, SSH or encryption keys and API tokens to interact with users and other services (Shaikh, 2016). They also need usernames and passwords to authenticate application users or data to access centralized authentication mechanisms. Many common techniques for storing secrets on dedicated hosts or virtual machines -- such as dotfiles with limited access or storage within encrypted files – aren't viable in container environments. When secrets are stored inside of an image, these secrets can be accessed by anyone that pulls the image from the registry (Shaikh, 2016). A commonly-proposed technique is to store secrets within environment variables used by the Docker containers (Mouat, 2015b). While preferable to storing within the image (Mouat, 2015b), secrets stored in environment variables are accessible to all container processes (Van Stijn, 2015). These secrets may then be leaked in logs (Mouat, 2015b), may be visible using Docker inspect commands, and may be shared with linked containers (Van Stijn, 2015). Even when application containers are re-deployed often, leaked credentials may linger, providing access to other, longer-lived services (Mouat, 2015b).

As an alternative to these approaches, secret stores such as Vault, Keywhiz, Crypt, or Sneaker allow secure storage, quick rotation of secrets, and auditability of secret access (Shaikh, 2016). These solutions are all somewhat immature, adding operational complexity to the overall solution, but are developing and maturing rapidly (Mouat, 2015b). Several have recently gone through independent security audits. Where images containing secrets need to be re-deployed to update (Shaikh, 2016), these tools allow rotation of credentials frequently and in some cases automatically. Cloud Foundry's Jason Smith noted that most attacks take 1) time, 2) leaked or misused credentials, and 3) misconfigured and/or unpatched software. By changing credentials frequently, we starve attackers of this required resource.

Checklist:

- ☐ Secrets are not stored in the Dockerfiles/included in the image
- ☐ Secrets are not stored in environment variables
- ☐ Secrets are not stored in volume mounts on the host
- ☐ Secrets are stored in a dedicated secrets management system
- ☐ Secrets are rotated frequently

5.6 Base Image Security

In addition to checking the provenance of images pulled from registries, Docker Administrators need to be able to trust that their images have up-to-date software, without any known vulnerabilities (Mouat, 2015b). When software is installed, package versions should be specified: this provides repeatable results and eliminates versions with known vulnerabilities (Docker, 2016). Similarly, signatures should be verified on any package downloads to ensure there is no tampering or corruption in transit. (Damato, 2014) Base images should be updated regularly with all security patches, with dependent images re-built and re-deployed, rather than updated in place (Petazzoni, 2015). To enable for this, the deployment team must be able to identify the list of images that are currently running and have a mechanism for rolling container restarts. By tagging images correctly in the registry, it is easy to track back to the build information and debug any issues (Mouat, 2015b). If Docker Hub is used to build images, they can be automatically rebuilt when the base image is updated (Docker, 2016).

Alyssa Robinson, lyssanr@yahoo.com

- ☐ Specify package versions and hash in FROM tag
- ☐ Understand the contents of running images
- ☐ Scan images regularly for vulnerabilities
- ☐ Update base image regularly and re-deploy containers

5.7 Limit container resources

As the number of containers running in a system grows, so does the likelihood that one of the applications has a security flaw that can lead to a denial of service for the other containers. If an attacker can monopolize a shared resource on one container, he may be able to deny access to multiple others (Mouat, 2015b). Docker provides the capability to limit the resources a container can use in order to give each a fair share while preventing starvation for the others. These limits can apply to memory, CPU, disk IOPS, plus processes and open file descriptors via container ulimits. Limits can also apply to the number of container restarts to keep problematic containers from using up host resources on restart (Docker, 2016). For CPU and block limits, these are relative limits, whereas for memory the limits are absolute (Goldmann, 2014).

In addition to limiting system resources, filesystem access can be limited by mounting filesystems read-only where possible; this prevents data overwrite, as well as injection of malicious code into the system (Walsh, 2015). Limiting filesystems to read-only can also provide benefits during an audit, removing the need to audit the running containers and leaving the focus on the image (Mouat, 2015b).

By default, containers can send network traffic using the internal Docker network whether or not ports are explicitly opened (Mouat, 2015b). This increases the attack surface and may provide an attacker the ability to pivot between containers; Docker users should turn off inter-container communication and require explicit linking of containers or opening of public ports for communication (CIS, 2016). While Docker containers are run with limited Linux capabilities by default, additional capabilities can be dropped or added as needed (Docker, 2016). While finding the right set can take a great deal of trial and error, limiting these capabilities provides further protection for the host in case of container breach (Mouat, 2015a).

Checklist:

- ☐ Mount filesystems read-only (prevent writing malicious code)
- ☐ Place limits on system resources (limit denial of service)
- ☐ Limit kernel calls (limit container breakout) and Linux capabilities
- ☐ Restrict network access (prevent attack pivot, data egress, -icc=false)

6. Tools for Docker Audit

As Docker grows in popularity and begins to move into production environment, auditors will need tools to determine whether containerized applications follow best practices. Ideally, DevOps teams will use these same tools as part of their deployment processes, stopping security issues before they ever make it into production (Raising the Floor Consortium, 2016). Current tools for Docker audit are immature, but evolving rapidly (de la Fuente, 2015). Among the best of these tools is Docker Bench for Security, which checks for compliance with the CIS Docker Benchmark (Mouat, 2015b. Docker Bench is both easy to run – launched from its own Docker container -- and easy to interpret the output, but does have limitations, including the fact that it isn't compatible with many of the common hardening guidelines. Batten, another CIS Benchmark scanner, is similarly easy to run but the output is somewhat less useful than Docker Bench for Security (de la Fuente, 2015). Banyan Collector, CoreOS Clair, and OpenSCAP Container Compliance can all check containers or images –either locally or within a registry -- for known vulnerabilities. OpenSCAP Container Compliance is supported only for RedHat/CentOS/Fedora but is highly customizable and simple to install and to use on these platforms (Raising the Floor Consortium, 2016). CoreOS Clair and Banyan Collector require additional work to install but provide additional coverage and Clair provides an API for process automation (CoreOS, 2016).

7. Conclusion

As use of Docker images in production becomes mainstream – a DevOps.com survey showed a 96% increase in the use of containers for production environments in 2016 (Ferranti, 2016) – auditors will need to be familiar with Docker best practices and audit tools. The development advantages of containers and their alignment to current DevOps and microservices deployment models make it likely that the growth in Docker use will continue for the foreseeable future (Kearns, 2016). With the rapid evolution of Docker and its surrounding tools, Docker security

audit is clearly a work in progress. Given the security advantages gained by incorporating audit tools into continuous deployment pipelines, (Bellis, 2015) as well as the potential pitfalls from ignoring Docker best practices, the time for creating checklists and understanding the available tools is now.

Alyssa Robinson, lyssanr@yahoo.com

8. References

Ankerholz, A. (2016, April 12). 8 Container Orchestration Tools to Know. Retrieved August 22, 2016, from https://www.linux.com/news/8-open-source-container-orchestration-tools-know

Banyanops. Banyanops/collector. *GitHub*. 2015. Retrieved October 6, 2016 from https://github.com/banyanops/collector/

Boelen, M. (2015, January 22). Docker Security: Best Practices for your Vessel & Containers. Retrieved August 27, 2016, from https://linux-audit.com/docker-security-best-practices-for-your-vessel-and-containers/

Boettiger, C. (2014, August 29). Docker tricks of the trade and best practices thoughts. Retrieved September 1, 2016, from http://www.carlboettiger.info/2014/08/29/docker-notes.html

Center for Internet Security. (2016). *CIS Docker 1.11.0 Benchmark* (Vol. 1.0.0, Publication). Retrieved from https://benchmarks.cisecurity.org/tools2/docker/CIS_Docker_1.11.0_Benchmark_v1.0.0.pdf

Chanezon, P. (2016, March 24). Docker for Mac and Windows Beta: The simplest way to use Docker on your laptop - Docker Blog. Retrieved August 20, 2016, from https://blog.docker.com/2016/03/docker-for-mac-windows-beta/

Close, M. (2015, October 15). Protecting Sensitive Information in Docker Container Images. Retrieved October 17, 2016, from https://www.ctl.io/developers/blog/post/tutorial-protecting-sensitive-info-docker

Cloud Foundry (2016, June 16). Hope Versus Reality: Containers in 2016. Retrieved August 15, 2016, from https://www.cloudfoundry.org/wp-content/uploads/2016/06/Cloud-Foundry-2016-Container-Report.pdf

Coreos. Coreos/clair. *GitHub*. N.p., Sept. 2016. Web. Retrieved October 3, 2016 from https://github.com/coreos/clair

Cramer, D. (2016, February 04). Got Docker? 4 Docker Management Tips. Retrieved August 18, 2016, from http://www.bmc.com/blogs/got-docker-4-docker-management-tips/

Damato, J. (2014, October 28). HOWTO: GPG sign and verify deb packages and APT repositories. Retrieved September 03, 2016, from http://blog.packagecloud.io/eng/2014/10/28/howto-gpg-sign-verify-deb-packages-apt-repositories/

Datadog. (2016, June). 8 surprising facts about real Docker adoption - Datadog. Retrieved September 1, 2016, from https://www.datadoghq.com/docker-adoption/

Alyssa Robinson, lyssanr@yahoo.com

De la Fuente, T. (2015, December 1). Docker Security Tools: Audit and Vulnerability Assessment. Retrieved September 15, 2016, from http://blyx.com/2015/12/01/docker-security-tools-audit-and-vulnerability-assessment/

Docker, Inc. (2016). Docker Overview. Retrieved August 2, 2016, from https://docs.docker.com/

Estes, P. (2015, October 13). User namespaces have arrived in Docker! Retrieved July 23, 2016, from https://integratedcode.us/2015/10/13/user-namespaces-have-arrived-in-docker/

Ferranti, Michael. "ClusterHQ." *Survey: 96% Increase in Container Production Usage over past Year* ·. N.p., 16 June 2016. Retrieved October 2, 2016 from https://clusterhq.com/2016/06/16/container-survey/

Flower, Z. (2015, September 22). Who Controls Docker Containers. Retrieved November 9, 2016, from https://www.sumologic.com/blog-devops/docker-containers/

Fowler, M., & Lewis, J. (2014, March 25). Microservices. Retrieved September 04, 2016, from http://martinfowler.com/articles/microservices.html

Goldmann, M. (2014, September 11). Resource management in Docker. Retrieved August 28, 2016, from https://goldmann.pl/blog/2014/09/11/resource-management-in-docker/

Greenbone Networks GmbH. "Greenbone Security Manager with Greenbone OS 3.1 - User Manual¶." *Greenbone Security Manager with Greenbone OS 3.1*. N.p., n.d. Retrieved October 5, 2016 from http://docs.greenbone.net/GSM-Manual/gos-3.1/en/operation.html

Gürkök, C. (2016, June 21). *Securing the Container Pipeline*. Speech presented at Dockercon 2016, Seattle, WA. Retrieved from https://blog.docker.com/2016/07/dockercon-2016-videos-use-case/

Haff, G. (2013, September 13). Connections: What are containers and how did they come about? Retrieved August 8, 2016, from http://bitmason.blogspot.com.au/2013/09/what-are-containers-anyway.html

Hausenblas, M. (2015, October 14). Docker Registries: The Good, the Bad & the Ugly. Retrieved August 22, 2016, from https://mesosphere.com/blog/2015/10/14/docker-registries-the-good-the-bad-the-ugly/

Hildred, T. (2015, August 28). The History of Containers. Retrieved August 16, 2016, from http://rhelblog.redhat.com/2015/08/28/the-history-of-containers/

Hoelzer, D. (2015). Auditing & Monitoring Networks, Perimeters & Systems. SANS Institute.

Humble, J. (2010). Introduction. Retrieved July 04, 2016, from https://continuousdelivery.com/

Alyssa Robinson, lyssanr@yahoo.com

Janetakis, N. (2016, May 02). Save Yourself from Years of Turmoil by Using Docker Today -. Retrieved August 08, 2016, from http://nickjanetakis.com/blog/save-yourself-from-years-of-turmoil-by-using-docker-today

Kearns, A. (2016, September 01). 3 facts about container adoption you don't know. Retrieved September 04, 2016, from https://www.oreilly.com/ideas/3-facts-about-container-adoption-you-dont-know?imm_mid=0e7611

Koohgoli, M . (2014, July 16). Tips for Tracking Open-Source Security Vulnerabilities - Law360. Retrieved October 16, 2016, from http://www.law360.com/articles/555153/tips-for-tracking-open-source-security-vulnerabilities

Kuusik, K. (2015, June 19). Docker Security – Admin Controls - Container Solutions. Retrieved September 03, 2016, from http://container-solutions.com/docker-security-admin-controls-2/

Linthicum, D. (2015, October). Securing Docker containers should top IT's to-do list. Retrieved August 25, 2016, from http://searchitoperations.techtarget.com/tip/Securing-Docker-containers-should-top-ITs-to-do-list

McGuire, K. (2015, August 17). The Truth About Docker Container Lifecycles. Retrieved July 5, 2016, from http://events.linuxfoundation.org/sites/events/files/slides/cc15_mcguire.pdf

Miell, I. (2016, July 08). A checklist for Docker in the Enterprise. Retrieved August 10, 2016, from https://zwischenzugs.wordpress.com/2016/07/08/a-checklist-for-docker-in-the-enterprise/

Mouat, A. (2014, February 17). 6 Dockerfile Tips from the Official Images - Container Solutions. Retrieved August 27, 2016, from http://container-solutions.com/6-dockerfile-tips-official-images/

Mouat, A. (2015a, October 20). Swarm v. Fleet v. Kubernetes v. Mesos. Retrieved August 22, 2016, from https://www.oreilly.com/ideas/swarm-v-fleet-v-kubernetes-v-mesos

Mouat, A. (2015b). *Using Docker*. Sebastopol, CA: O'Reilly Media.

MSV, J. (2016, May 11). From Containers to Container Orchestration - The New Stack. Retrieved August 22, 2016, from http://thenewstack.io/containers-container-orchestration/

Newman, A. (2015, December 14). Top 5 Docker Logging Methods to Fit Your Container Deployment Strategy. Retrieved August 23, 2016, from https://www.loggly.com/blog/top-5-docker-logging-methods-to-fit-your-container-deployment-strategy/

Nickeloff, J. (2015, February 03). Hardening Docker Containers: Disable SUID Programs. Retrieved September 1, 2016, from https://blog.tutum.co/2015/02/03/hardening-containers-disable-suid-programs/

Alyssa Robinson, lyssanr@yahoo.com

Nieto, C. (2015, April 2). Reventlov's silly hacks. Retrieved August 27, 2016, from http://reventlov.com/advisories/using-the-docker-command-to-root-the-host

Open Source Initiative. Frequently Answered Questions. (n.d.). Retrieved October 13, 2016, from https://opensource.org/faq

OWASP. (2016, January 20). Logging Cheat Sheet. Retrieved September 23, 2016, from https://www.owasp.org/index.php/Logging_Cheat_Sheet

Petazzoni, J. (2015, May 27). Someone said that 30% of the images on the Docker Registry contain vulnerabilities. Retrieved August 16, 2016, from https://jpetazzo.github.io/2015/05/27/docker-images-vulnerabilities/

Puppet Labs, & DORA. (2016). State of DevOps Report. Retrieved August 8, 2016, from https://puppet.com/system/files/2016-06/2016 State of DevOps Report_0.pdf

Raising the Floor Consortium. "Vulnerability Assessment." *GPII*. N.p., 7 July 2016. Web. 15 Sept. 2016.

Shaikh, I. (2016, January 22). Runtime secrets with Docker containers. Retrieved August 23, 2016, from http://elasticcompute.io/2016/01/21/runtime-secrets-with-docker-containers/

Smith, J. (2016, April 19). The Three R's of Enterprise Security: Rotate, Repave, and Repair Retrieved April 27, 2016 from https://medium.com/built-to-adapt/the-three-r-s-of-enterprise-security-rotate-repave-and-repair-f64f6d6ba29d?imm_mid=0e30ba&cmp=em-na-na-na-newsltr_security_20160426#.8780qrvlt

Van Stijn, S. (2015, May 26). Secrets: Write-up best practices, do's and don'ts, roadmap · Issue #13490 · docker/docker. Retrieved July 22, 2016, from https://github.com/docker/docker/issues/13490

Walsh, D. (2014, July 22). Are Docker containers really secure? Retrieved July 07, 2016, from https://opensource.com/business/14/7/docker-security-selinux

Wang, C. (2016, May 26). Containers 101: Linux containers and Docker explained. Retrieved August 20, 2016, from http://www.infoworld.com/article/3072929/linux/containers-101-linux-containers-and-docker-explained.html

Weins, K. (2016, February 26). Cloud Computing Trends: 2016 State of the Cloud Survey. Retrieved July 05, 2016, from http://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2016-state-cloud-survey

World Shipping Council - Partners in Trade. (2016). Retrieved August 16, 2016, from http://www.worldshipping.org/about-the-industry/history-of-containerization

Alyssa Robinson, lyssanr@yahoo.com

Appendix A: Audit Checklist

1. **Ensure good host security**
   - ☐ Patch hosts
   - ☐ Install only needed components
   - ☐ Hardened kernel (Grsecurity, PaX)
   - ☐ AppArmor/SElinux
   - ☐ Segregate containers by data sensitivity – requires manual audit unless images can be named or tagged with sensitivity levels
   - ☐ Update Docker software regularly

2. **Check Image Provenance**
   - ☐ Enable DOCKER_CONTENT_TRUST environment variable, so that only signed images can be pulled.
   - ☐ Store root key offline – manual/process check required
   - ☐ Back up signing keys; rotate & expire old keys – manual/process check required
   - ☐ Check image digest at deployment
3. **Monitor Containers**
   - ☐ Capture host logs
   - ☐ Capture logs from Docker infrastructure
   - ☐ Capture container logs
   - ☐ Ensure adequate log information at the containers
     - Ensure –log-level is set to INFO (default) (CIS, 2016)
4. **Do Not Run Container Processes as Root**
   - ☐ Container processes run as non-privileged USER
   - ☐ If process must run as root, use Docker User Namespace feature to re-map to non-privileged user on host
   - ☐ Limit access to run Docker
   - ☐ Remove setUID/setGID binaries
5. **Do Not Store Secrets in Containers**
   - ☐ Secrets are not stored in the Dockerfiles/included in the image
   - ☐ Secrets are not stored in environment variables

Alyssa Robinson, lyssanr@yahoo.com

- ☐ Secrets are not stored in volume mounts on the host
- ☐ Secrets are stored in a dedicated secrets management system
- ☐ Secrets are rotated frequently

6. **Base Image Security**
   - ☐ Specify package versions and hash in FROM tag
   - ☐ Understand the contents of running images
   - ☐ Scan images regularly for vulnerabilities
   - ☐ Update base image regularly and re-deploy containers

7. **Limit container resources**
   - ☐ Mount filesystems read-only (prevent writing malicious code)
   - ☐ Place limits on system resources (limit denial of service)
   - ☐ Limit kernel calls (limit container breakout) and Linux capabilities
   - ☐ Restrict network access (prevent attack pivot, data egress, -icc=false)

Appendix B: Audit Tools Step by Step

1.Ensure good host security

1.1 Checklist:

- ☐ Patch hosts
- ☐ Install only needed components
- ☐ Hardened kernel (Grsecurity, PaX)
- ☐ AppArmor/SElinux
- ☐ Segregate containers by data sensitivity – requires manual audit unless images can be named or tagged with sensitivity levels
- ☐ Update Docker software regularly

1.2 OpenVAS Scan showing host vulnerabilities:

OpenVAS is an open source scanner that does both external and authenticated scans. It uses a CVE feed as well as compliance scans for auditing:

Alyssa Robinson, lyssanr@yahoo.com

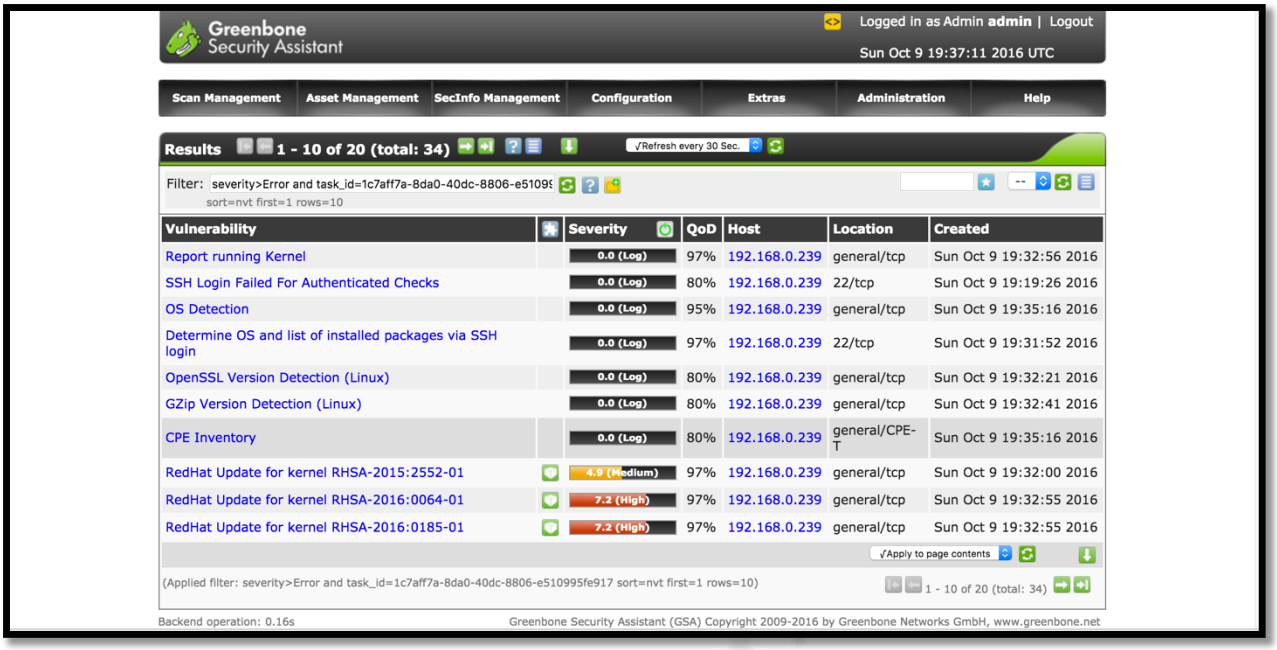


Figure 3 OpenVAS scan showing unpatched vulnerabilities

1.3 Running Batten scan to check for non-essential services running:

Batten checks Docker and host configuration against the Docker CIS Benchmark. It is automated and easy to run, but the output is not as useful as Docker Bench.

```
ubuntu@ip-192-168-0-235:~$ docker run -v
/var/run/docker.sock:/var/run/docker.sock jerbi/batten
[1/68] FAILED [CIS-Docker-Benchmark-1.1] Create a separate
partition for containers
  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.
  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.
[2/68] PASSED [CIS-Docker-Benchmark-1.2] Use the updated Linux
Kernel
[3/68] PASSED [CIS-Docker-Benchmark-1.3] Do not use development
tools in production
[4/68] PASSED [CIS-Docker-Benchmark-1.4] Harden the container host
[5/68] PASSED [CIS-Docker-Benchmark-1.5] Remove all non-essential
services from the host
[6/68] PASSED [CIS-Docker-Benchmark-1.6] Keep Docker up to date
[7/68] PASSED [CIS-Docker-Benchmark-1.7] Only allow trusted users
to control Docker daemon
```

Alyssa Robinson, lyssanr@yahoo.com

1.4 CIS-CAT Scan after SELinux configured with Docker Profile:

```
[ec2-user@ip-192-168-0-239 ~]$ sudo sestatus
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             targeted
Current mode:                   enforcing
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Max kernel policy version:      28
```

Summary

Description	Tests				Scoring		
	Pass	Fail	Error	Unkn.	Score	Max	Percent
1 Initial Setup	**28**	**22**	**0**	**0**	**25.0**	**44.0**	**57%**
1.1 Filesystem Configuration	11	15	0	0	11.0	26.0	42%
1.1.1 Disable unused filesystems	0	8	0	0	0.0	8.0	0%
1.2 Configure Software Updates	1	0	0	0	1.0	1.0	100%
1.3 Filesystem Integrity Checking	0	2	0	0	0.0	2.0	0%
1.4 Secure Boot Settings	1	2	0	0	1.0	2.0	50%
1.5 Additional Process Hardening	3	1	0	0	2.0	3.0	67%
1.6 Mandatory Access Control	7	0	0	0	7.0	7.0	100%
1.6.1 Configure SELinux	6	0	0	0	6.0	6.0	100%
1.7 Warning Banners	5	2	0	0	3.0	3.0	100%
1.7.1 Command Line Warning Banners	4	2	0	0	2.0	2.0	100%
2 Services	**34**	**1**	**0**	**0**	**33.0**	**34.0**	**97%**
2.1 inetd Services	7	0	0	0	7.0	7.0	100%
2.2 Special Purpose Services	22	1	0	0	21.0	22.0	95%
2.2.1 Time Synchronization	3	0	0	0	2.0	2.0	100%
2.3 Service Clients	5	0	0	0	5.0	5.0	100%
3 Network Configuration	**10**	**16**	**0**	**0**	**10.0**	**21.0**	**48%**
3.1 Network Parameters (Host Only)	0	2	0	0	0.0	2.0	0%
3.2 Network Parameters (Host and Router)	6	2	0	0	6.0	8.0	75%
3.3 IPv6	0	3	0	0	0.0	2.0	0%
3.4 TCP Wrappers	3	2	0	0	3.0	5.0	60%
3.5 Uncommon Network Protocols	0	4	0	0	0.0	0.0	0%
3.6 Firewall Configuration	1	3	0	0	1.0	4.0	25%
4 Logging and Auditing	**6**	**21**	**0**	**0**	**5.0**	**26.0**	**19%**

Figure 4 CIS-CAT Scan output showing SElinux enabled

1.5 Docker Bench for Security, showing updated Docker versions:

```
>     docker/docker-bench-security
# ------------------------------------------------------------------------
-------------
# Docker Bench for Security v1.1.0
#
# Docker, Inc. (c) 2015-
#
# Checks for dozens of common best practices around deploying
Docker containers in production.
# Inspired by the CIS Docker 1.11 Benchmark:
```

Alyssa Robinson, lyssanr@yahoo.com

```
# https://benchmarks.cisecurity.org/downloads/show-
single/index.cfm?file=docker16.110
# ------------------------------------------------------------
------------

Initializing Sun Oct  9 21:02:45 UTC 2016


[INFO] 1 - Host Configuration
[WARN] 1.1  - Create a separate partition for containers
[PASS] 1.2  - Use an updated Linux Kernel
[PASS] 1.4  - Remove all non-essential services from the host -
Network
[PASS] 1.5  - Keep Docker up to date
[INFO]       * Using 1.12.1 which is current as of 2016-08-16
[INFO]       * Check with your operating system vendor for support
and security maintenance for docker
[INFO] 1.6  - Only allow trusted users to control Docker daemon
```

2. Check Image Provenance

2.1 Checklist:

- ☐ Enable DOCKER_CONTENT_TRUST environment variable, so that only signed images can be pulled.
- ☐ Store root key offline – manual/process check required
- ☐ Back up signing keys; rotate & expire old keys – manual/process check required
- ☐ Check image digest at deployment

2.2 Auditing Docker Content Trust:

Docker Content Trust can be enabled via environment variables or via the docker run command line.

Checking environment variables:

```
ubuntu@ip-192-168-0-235:~$ env
XDG_SESSION_ID=12
TERM=xterm-256color
SHELL=/bin/bash
USER=ubuntu
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35
:bd=40;33;01:cd=40;33;01:or=40;31;01:su=37;41:sg=30;43:ca=30;41:t
w=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arj=
01;31:*.taz=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;3
1:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lz=01;3
1:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz
=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;3
1:*.sar=01;31:*.rar=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.
7z=01;31:*.rz=01;31:*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bmp=01
;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*
```

Alyssa Robinson, lyssanr@yahoo.com

```
.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svg
z=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01
;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:
*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv
=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;3
5:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd
=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.axv=01;35:*.anx=01;3
5:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m
id=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=0
0;36:*.ra=00;36:*.wav=00;36:*.axa=00;36:*.oga=00;36:*.spx=00;36:*
.xspf=00;36:
MAIL=/var/mail/ubuntu
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
:/usr/games:/usr/local/games
PWD=/home/ubuntu
LANG=en_US.UTF-8
SHLVL=1
HOME=/home/ubuntu
LOGNAME=ubuntu
DOCKER_CONTENT_TRUST=1
SSH_CONNECTION=198.135.0.233 40978 192.168.0.235 22
LESSOPEN=| /usr/bin/lesspipe %s
XDG_RUNTIME_DIR=/run/user/1000
LESSCLOSE=/usr/bin/lesspipe %s %s
_=/usr/bin/env
```

Checking command line:

```
ubuntu@ip-192-168-0-235:~$ docker run -m 256m -c 512 --name some-
nginx --read-only -v /var/cache/nginx -v /var/run -v
/dev/log:/dev/log --cap-drop SETUID --cap-drop SETGID --security-
opt="no-new-privileges" -restart on-failure:5 --disable-content-
trust=false -d nginx
```

3. Monitor Containers

3.1 Checklist:

- ☐ Capture host logs
- ☐ Capture logs from Docker infrastructure
- ☐ Capture container logs
- ☐ Ensure adequate log information at the containers
    - Ensure –log-level is set to INFO (default) (CIS, 2016)
- ☐ Role-based monitoring

3.2 CIS-CAT Scan showing centralized logging for host:

```
[ec2-user@ip-192-168-0-239 cis-cat-full]$ sudo ./CIS-CAT.sh --find
This is CIS-CAT version 3.0.29
```

Alyssa Robinson, lyssanr@yahoo.com

```
#1 -- CIS AIX 4.3.2, 4.3.3, 5L, 5.1 Benchmark
#2 -- CIS AIX 5.3 and AIX 6.1 Benchmark
#3 -- CIS Amazon Linux Benchmark
#4 -- CIS Apache Tomcat 5.5 and 6.0 Benchmark
#5 -- CIS Apple OSX 10.10 Benchmark
#6 -- CIS Apple OSX 10.11 Benchmark
#7 -- CIS Apple OSX 10.5 Benchmark
#8 -- CIS Apple OSX 10.6 Benchmark
#9 -- CIS Apple OSX 10.8 Benchmark
#10 -- CIS Apple OSX 10.9 Benchmark
#11 -- CIS CentOS Linux 6 Benchmark
#12 -- CIS CentOS Linux 7 Benchmark
#13 -- CIS Cisco Firewall Benchmark
#14 -- CIS Cisco IOS 12 Benchmark
#15 -- CIS Cisco IOS 15 Benchmark
#16 -- CIS Debian Linux 3 Benchmark
#17 -- CIS Debian Linux 7 Benchmark
#18 -- CIS Debian Linux 8 Benchmark
#19 -- CIS Google Chrome Benchmark
#20 -- CIS HP-UX 11i v3 Update 2 Benchmark
#21 -- CIS IBM AIX 7.1 Benchmark
#22 -- CIS MIT Kerberos 1.10 Benchmark
#23 -- CIS Microsoft IIS 7 Benchmark
#24 -- CIS Microsoft IIS 8 Benchmark
#25 -- CIS Microsoft Internet Explorer 10 Benchmark
#26 -- CIS Microsoft Internet Explorer 11 Benchmark
#27 -- CIS Microsoft Office 2013 Benchmark
#28 -- CIS Microsoft Office 2016 Benchmark
#29 -- CIS Microsoft Office Access 2013 Benchmark
#30 -- CIS Microsoft Office Access 2016 Benchmark
#31 -- CIS Microsoft Office Excel 2013 Benchmark
#32 -- CIS Microsoft Office Excel 2016 Benchmark
#33 -- CIS Microsoft Office Outlook 2013 Benchmark
#34 -- CIS Microsoft Office Outlook 2016 Benchmark
#35 -- CIS Microsoft Office PowerPoint 2013 Benchmark
#36 -- CIS Microsoft Office PowerPoint 2016 Benchmark
#37 -- CIS Microsoft Office Word 2013 Benchmark
#38 -- CIS Microsoft Office Word 2016 Benchmark
#39 -- CIS Microsoft SQL Server 2008 R2 Benchmark
#40 -- CIS Microsoft SQL Server 2012 Benchmark
#41 -- CIS Microsoft SQL Server 2014 Benchmark
#42 -- CIS Microsoft Windows 10 Enterprise (Release 1511) Benchmark
#43 -- CIS Microsoft Windows 7 Benchmark
#44 -- CIS Microsoft Windows 8 Benchmark
#45 -- CIS Microsoft Windows 8.1 Benchmark
#46 -- CIS Microsoft Windows Server 2003 Benchmark
#47 -- CIS Microsoft Windows Server 2008 (non-R2) Benchmark
#48 -- CIS Microsoft Windows Server 2008 R2 Benchmark
#49 -- CIS Microsoft Windows Server 2012 (non-R2) Benchmark
#50 -- CIS Microsoft Windows Server 2012 R2 Benchmark
```

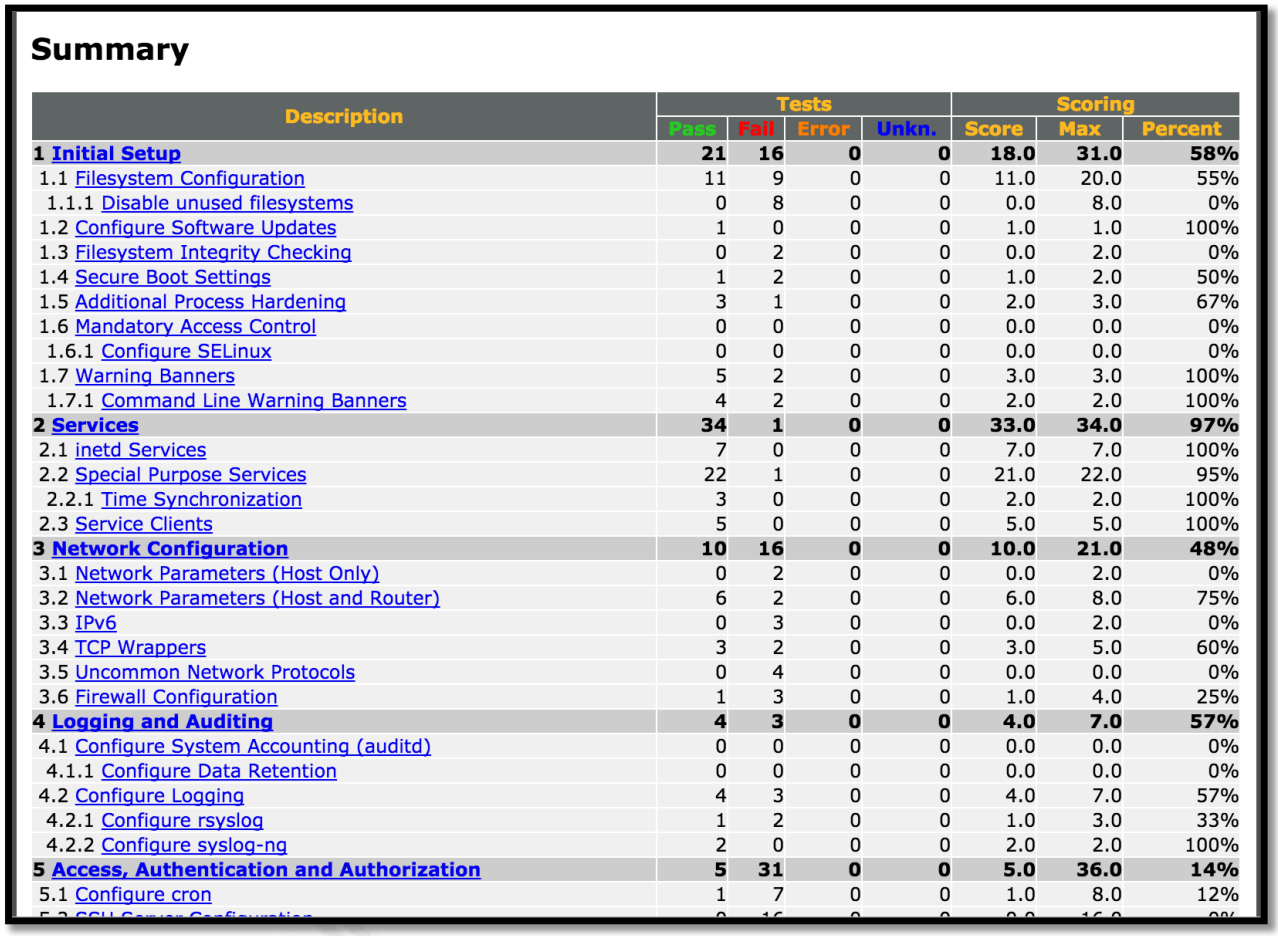
Alyssa Robinson, lyssanr@yahoo.com

```
#51 -- CIS Microsoft Windows XP Benchmark
#52 -- CIS Mozilla Firefox 24 ESR Benchmark
#53 -- CIS Mozilla Firefox 3 Benchmark
#54 -- CIS Mozilla Firefox 38 ESR Benchmark
#55 -- CIS Oracle Database 10g Benchmark
#56 -- CIS Oracle Database 11g Benchmark
#57 -- CIS Oracle Database 11g R2 Benchmark
#58 -- CIS Oracle Database 12c Benchmark
#59 -- CIS Oracle Linux 6 Benchmark
#60 -- CIS Oracle Linux 7 Benchmark
#61 -- CIS Oracle MySQL Community Server 5.6 Benchmark
#62 -- CIS Oracle MySQL Community Server 5.7 Benchmark
#63 -- CIS Oracle MySQL Enterprise Edition 5.6 Benchmark
#64 -- CIS Oracle MySQL Enterprise Edition 5.7 Benchmark
#65 -- CIS Oracle Solaris 10 Benchmark
#66 -- CIS Oracle Solaris 11 Benchmark
#67 -- CIS Oracle Solaris 11.1 Benchmark
#68 -- CIS Oracle Solaris 11.2 Benchmark
#69 -- CIS Oracle Solaris 2.5.1 - 9 Benchmark
#70 -- CIS Red Hat 4 and Fedora Core 1, 2, 3, 4, 5 Benchmark
#71 -- CIS Red Hat Enterprise Linux 5 Benchmark
#72 -- CIS Red Hat Enterprise Linux 6 Benchmark
#73 -- CIS Red Hat Enterprise Linux 7 Benchmark
#74 -- CIS SUSE Linux Enterprise Server 11 Benchmark
#75 -- CIS SUSE Linux Enterprise Server 12 Benchmark
#76 -- CIS Slackware 10.2 Linux Benchmark
#77 -- CIS SuSE Linux Enterprise Server 10.0 SP1 Benchmark
#78 -- CIS SuSE Linux Enterprise Server 9.0 Benchmark
#79 -- CIS Ubuntu 12.04 LTS Server Benchmark
#80 -- CIS Ubuntu Linux 14.04 LTS Benchmark
#81 -- CIS Ubuntu Linux 16.04 LTS Benchmark
#82 -- CIS VMware ESX 3.5 Benchmark
#83 -- CIS VMware ESX 4 Benchmark
#84 -- CIS VMware ESXi 5.5 Benchmark
Which benchmark should be used? (return to exit)
73
Selected CIS Red Hat Enterprise Linux 7 Benchmark

#1 -- xccdf_org.cisecurity.benchmarks_profile_Level_1_-_Server
#2 -- xccdf_org.cisecurity.benchmarks_profile_Level_2_-_Server
#3 -- xccdf_org.cisecurity.benchmarks_profile_Level_1_-_Workstation
#4 -- xccdf_org.cisecurity.benchmarks_profile_Level_2_-_Workstation
Which Profile should be used? (return to exit)
1
Selected Level 1 - Server

Add another benchmark? (Y/n/q)
n
```

```
HTML Report written to: /root/CIS-CAT_Results/ip-192-168-0-
239.ec2.internal-20161007T004638Z/ip-192-168-0-239.ec2.internal-
report-20161007T004638Z.html
```



Summary

Description	Tests				Scoring		
	Pass	Fail	Error	Unkn.	Score	Max	Percent
1 Initial Setup	**21**	**16**	**0**	**0**	**18.0**	**31.0**	**58%**
1.1 Filesystem Configuration	11	9	0	0	11.0	20.0	55%
1.1.1 Disable unused filesystems	0	8	0	0	0.0	8.0	0%
1.2 Configure Software Updates	1	0	0	0	1.0	1.0	100%
1.3 Filesystem Integrity Checking	0	2	0	0	0.0	2.0	0%
1.4 Secure Boot Settings	1	2	0	0	1.0	2.0	50%
1.5 Additional Process Hardening	3	1	0	0	2.0	3.0	67%
1.6 Mandatory Access Control	0	0	0	0	0.0	0.0	0%
1.6.1 Configure SELinux	0	0	0	0	0.0	0.0	0%
1.7 Warning Banners	5	2	0	0	3.0	3.0	100%
1.7.1 Command Line Warning Banners	4	2	0	0	2.0	2.0	100%
2 Services	**34**	**1**	**0**	**0**	**33.0**	**34.0**	**97%**
2.1 inetd Services	7	0	0	0	7.0	7.0	100%
2.2 Special Purpose Services	22	1	0	0	21.0	22.0	95%
2.2.1 Time Synchronization	3	0	0	0	2.0	2.0	100%
2.3 Service Clients	5	0	0	0	5.0	5.0	100%
3 Network Configuration	**10**	**16**	**0**	**0**	**10.0**	**21.0**	**48%**
3.1 Network Parameters (Host Only)	0	2	0	0	0.0	2.0	0%
3.2 Network Parameters (Host and Router)	6	2	0	0	6.0	8.0	75%
3.3 IPv6	0	3	0	0	0.0	2.0	0%
3.4 TCP Wrappers	3	2	0	0	3.0	5.0	60%
3.5 Uncommon Network Protocols	0	4	0	0	0.0	0.0	0%
3.6 Firewall Configuration	1	3	0	0	1.0	4.0	25%
4 Logging and Auditing	**4**	**3**	**0**	**0**	**4.0**	**7.0**	**57%**
4.1 Configure System Accounting (auditd)	0	0	0	0	0.0	0.0	0%
4.1.1 Configure Data Retention	0	0	0	0	0.0	0.0	0%
4.2 Configure Logging	4	3	0	0	4.0	7.0	57%
4.2.1 Configure rsyslog	1	2	0	0	1.0	3.0	33%
4.2.2 Configure syslog-ng	2	0	0	0	2.0	2.0	100%
5 Access, Authentication and Authorization	**5**	**31**	**0**	**0**	**5.0**	**36.0**	**14%**
5.1 Configure cron	1	7	0	0	1.0	8.0	12%

Figure 5 CIS-CAT Scan output showing logging configuration

3.3 Docker Bench Check for logging configuration:

```
ubuntu@ip-192-168-0-235:/etc/default$ docker run -m 256m -c 512 --name
log-nginx --log-driver=syslog --read-only -v /var/cache/nginx -v
/var/run -v /dev/log:/dev/log --cap-drop SETUID --cap-drop SETGID --
security-opt="no-new-privileges" --restart on-failure:5  --log-opt
syslog-address=udp://192.168.0.235:514 --disable-content-trust=false -
d nginx

[INFO] 2 - Docker Daemon Configuration
[WARN] 2.1  - Restrict network traffic between containers
[PASS] 2.2  - Set the logging level
[PASS] 2.3  - Allow Docker to make changes to iptables
[PASS] 2.4  - Do not use insecure registries
[WARN] 2.5  - Do not use the aufs storage driver
[INFO] 2.6  - Configure TLS authentication for Docker daemon
[INFO]      * Docker daemon not listening on TCP
```

Alyssa Robinson, lyssanr@yahoo.com

```
[INFO] 2.7 - Set default ulimit as appropriate
[INFO]      * Default ulimit doesn't appear to be set
[WARN] 2.8  - Enable user namespace support
[PASS] 2.9  - Confirm default cgroup usage
[PASS] 2.10 - Do not change base device size until needed
[WARN] 2.11 - Use authorization plugin
[PASS] 2.12 - Configure centralized and remote logging
[WARN] 2.13 - Disable operations on legacy registry (v1)
```

4. Do Not Run Container Processes as Root

4.1 Checklist:

- ☐ Container processes run as non-privileged USER
- ☐ If process must run as root, use Docker User Namespace feature to re-map to non-privileged user on host
- ☐ Limit access to run Docker
- ☐ Remove setUID/setGID binaries

4.2 Example: Mongodb container running as root user

```
ubuntu@ip:~/docker_transcripts$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS
NAMES
df97c0c5c9b7        tutum/mongodb       "/run.sh"           22 seconds
ago      Up 20 seconds       0.0.0.0:27017->27017/tcp, 0.0.0.0:28017-
>28017/tcp   drunk_williams
ubuntu@ip:~/docker_transcripts$ docker exec -it drunk_williams
/bin/bash
root@df97c0c5c9b7:/# ps -deaf |grep mongo
root         5     1  0 23:13 ?        00:00:00 mongod --httpinterface
--rest --master --auth

ubuntu@ip-:~/docker_transcripts$ docker run -it --net host --pid host
--cap-add audit_control \
>     -v /var/lib:/var/lib \
>     -v /var/run/docker.sock:/var/run/docker.sock \
>     -v /usr/lib/systemd:/usr/lib/systemd \
>     -v /etc:/etc --label docker_bench_security \
>     docker/docker-bench-security
# ------------------------------------------------------------------------
----------
# Docker Bench for Security v1.1.0
#
# Docker, Inc. (c) 2015-
```

Alyssa Robinson, lyssanr@yahoo.com

```
#
# Checks for dozens of common best-practices around deploying Docker
containers in production.
# Inspired by the CIS Docker 1.11 Benchmark:
# https://benchmarks.cisecurity.org/downloads/show-
single/index.cfm?file=docker16.110
# ------------------------------------------------------------------
----------

[INFO] 4 - Container Images and Build Files
[WARN] 4.1  - Create a user for the container
[WARN]      * Running as root: drunk_williams
[WARN] 4.5  - Enable Content trust for Docker
```

4.3 Example: Mongodb running as non-root user

```
ubuntu@ip:~/docker_transcripts$  docker run -u mongodb -d -p
27017:27017 -p 28017:28017 tutum/mongodb
e907a05054f1d405502b3c297a757804bc9036094a5e79144c48240ae47f9243
ubuntu@ip:~/docker_transcripts$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS
NAMES
e907a05054f1        tutum/mongodb       "/run.sh"           3 seconds
ago       Up 2 seconds        0.0.0.0:27017->27017/tcp, 0.0.0.0:28017-
>28017/tcp   condescending_ritchie
ubuntu@ip:~/docker_transcripts$ docker exec -it condescending_ritchie
/bin/bash
mongodb@e907a05054f1:/$ ps -deaf |grep mongodb
mongodb      1     0  0 23:18 ?        00:00:00 /bin/bash /run.sh
mongodb      6     1  0 23:18 ?        00:00:00 /bin/bash
/set_mongodb_password.sh
mongodb     28     0  0 23:18 ?        00:00:00 /bin/bash
mongodb     48     6  0 23:18 ?        00:00:00 sleep 5
mongodb     49    28  0 23:18 ?        00:00:00 ps -deaf

ubuntu@ip-192-168-0-235:~/docker_transcripts$ docker run -it --net
host --pid host --cap-add audit_control     -v /var/lib:/var/lib     -
v /var/run/docker.sock:/var/run/docker.sock     -v
/usr/lib/systemd:/usr/lib/systemd     -v /etc:/etc --label
docker_bench_security     docker/docker-bench-security
# ------------------------------------------------------------------
----------
# Docker Bench for Security v1.1.0
#
# Docker, Inc. (c) 2015-
#
# Checks for dozens of common best-practices around deploying Docker
containers in production.
# Inspired by the CIS Docker 1.11 Benchmark:
```

```
# https://benchmarks.cisecurity.org/downloads/show-
single/index.cfm?file=docker16.110
# ------------------------------------------------------------
----------
[INFO] 4 - Container Images and Build Files
[PASS] 4.1  - Create a user for the container
```

4.4 Nginx container running with User Namespace mapping:

```
ubuntu@ip-192-168-0-235:/etc/default$ docker run -m 256m -c 512 --name
log-nginx --log-driver=syslog  -v /var/cache/nginx -v /var/run -v
/dev/log:/dev/log --cap-drop SETUID --cap-drop SETGID --security-
opt="no-new-privileges" --restart on-failure:5  --log-opt syslog-
address=udp://192.168.0.235:514 --disable-content-trust=false -d nginx

Note that Docker Bench cannot be run as a container with user
namespaces enabled:

ubuntu@ip-192-168-0-235:/etc/default$ docker run -it --net host --pid
host --cap-add audit_control     -v /var/lib:/var/lib     -v
/var/run/docker.sock:/var/run/docker.sock     -v
/usr/lib/systemd:/usr/lib/systemd     -v /etc:/etc --label
docker_bench_security     docker/docker-bench-security
docker: Error response from daemon: Cannot share the host's network
namespace when user namespaces are enabled.

ubuntu@ip-192-168-0-235:~/docker-bench-security$ sudo ./docker-bench-
security.sh
# ------------------------------------------------------------
----------
# Docker Bench for Security v1.1.0
#
# Docker, Inc. (c) 2015-
#
# Checks for dozens of common best-practices around deploying Docker
containers in production.
# Inspired by the CIS Docker 1.11 Benchmark:
# https://benchmarks.cisecurity.org/downloads/show-
single/index.cfm?file=docker16.110
# ------------------------------------------------------------
----------

[INFO] 2 - Docker Daemon Configuration
[WARN] 2.1  - Restrict network traffic between containers
[PASS] 2.2  - Set the logging level
[PASS] 2.3  - Allow Docker to make changes to iptables
[PASS] 2.4  - Do not use insecure registries
[WARN] 2.5  - Do not use the aufs storage driver
[INFO] 2.6  - Configure TLS authentication for Docker daemon
[INFO]      * Docker daemon not listening on TCP
[INFO] 2.7 - Set default ulimit as appropriate
```

```
[INFO]      * Default ulimit doesn't appear to be set
[PASS] 2.8  - Enable user namespace support
[PASS] 2.9  - Confirm default cgroup usage
[PASS] 2.10 - Do not change base device size until needed
[WARN] 2.11 - Use authorization plugin
[PASS] 2.12 - Configure centralized and remote logging
[WARN] 2.13 - Disable operations on legacy registry (v1)
```

4.5 Docker Bench checks regarding users with access to Docker daemon:

```
ubuntu@ip-192-168-0-235:~/docker_transcripts$ docker run -it --net
host --pid host --cap-add audit_control     -v /var/lib:/var/lib
-v /var/run/docker.sock:/var/run/docker.sock     -v
/usr/lib/systemd:/usr/lib/systemd     -v /etc:/etc --label
docker_bench_security     docker/docker-bench-security
# ------------------------------------------------------------------
-------------
# Docker Bench for Security v1.1.0
#
# Docker, Inc. (c) 2015-
#
# Checks for dozens of common best-practices around deploying
Docker containers in production.
# Inspired by the CIS Docker 1.11 Benchmark:
# https://benchmarks.cisecurity.org/downloads/show-
single/index.cfm?file=docker16.110
# ------------------------------------------------------------------
-------------
 [INFO] 1 - Host Configuration
[INFO] 1.6  - Only allow trusted users to control Docker daemon
[INFO]      * docker:x:999:ubuntu
```

5. Do Not Store Secrets in Containers

5.1 Checklist:

- ☐ Secrets are not stored in the Dockerfiles/included in the image
- ☐ Secrets are not stored in environment variables
- ☐ Secrets are not stored in volume mounts on the host
- ☐ Are stored in a secrets store
- ☐ Secrets are rotated frequently

5.2 Docker inspect shows secrets stored in environment variables:

```
ubuntu@ip-192-168-0-235:~$ docker images
REPOSITORY                    TAG                 IMAGE ID
CREATED             SIZE
```

```
      nginx                           latest              ba6bed934df2
      13 days ago         181.4 MB
      mysql                           latest              18f13d72f7f0
      13 days ago         383.4 MB
      docker/docker-bench-security    latest              2d4c144b12a4
      6 weeks ago         40.58 MB
      quay.io/coreos/clair            v1.2.2              4c58038d597f
      4 months ago        827.1 MB
      jerbi/batten                    latest              8e6c0e7cf775
      15 months ago       539.1 MB
      ubuntu@ip-192-168-0-235:~$ docker ps
      CONTAINER ID        IMAGE               COMMAND
      CREATED              STATUS              PORTS
      NAMES
ubuntu@ip-192-168-0-235:~$ docker inspect b7eb67dda521
[
    {
        "Id":
"b7eb67dda52101520968494fed45ad0fa6636cb8cbd7652ca79bfd85d51e1306",
        "Created": "2016-10-07T12:47:35.657229619Z",
        "Path": "docker-entrypoint.sh",
        "Args": [
            "mysqld"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 682,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2016-10-07T12:47:35.928088446Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image":
"sha256:18f13d72f7f0e105c09e78a9e44b194e91de05e13db93eec39f61fc6e95cd2
94",
        "ResolvConfPath":
"/var/lib/docker/containers/b7eb67dda52101520968494fed45ad0fa6636cb8cb
d7652ca79bfd85d51e1306/resolv.conf",
        "HostnamePath":
"/var/lib/docker/containers/b7eb67dda52101520968494fed45ad0fa6636cb8cb
d7652ca79bfd85d51e1306/hostname",
        "HostsPath":
"/var/lib/docker/containers/b7eb67dda52101520968494fed45ad0fa6636cb8cb
d7652ca79bfd85d51e1306/hosts",
        "LogPath":
```

```
"/var/lib/docker/containers/b7eb67dda52101520968494fed45ad0fa6636cb8cb
d7652ca79bfd85d51e1306/b7eb67dda52101520968494fed45ad0fa6636cb8cbd7652
ca79bfd85d51e1306-json.log",
        "Name": "/some-mysql",
        "RestartCount": 0,
        "Driver": "aufs",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "",
        "ExecIDs": null,
        "HostConfig": {
            "Binds": null,
            "ContainerIDFile": "",
            "LogConfig": {
                "Type": "json-file",
                "Config": {}
            },
            "NetworkMode": "default",
            "PortBindings": {},
            "RestartPolicy": {
                "Name": "no",
                "MaximumRetryCount": 0
            },
            "AutoRemove": false,
            "VolumeDriver": "",
            "VolumesFrom": null,
            "CapAdd": null,
            "CapDrop": null,
            "Dns": [],
            "DnsOptions": [],
            "DnsSearch": [],
            "ExtraHosts": null,
            "GroupAdd": null,
            "IpcMode": "",
            "Cgroup": "",
            "Links": null,
            "OomScoreAdj": 0,
            "PidMode": "",
            "Privileged": false,
            "PublishAllPorts": false,
            "ReadonlyRootfs": false,
            "SecurityOpt": null,
            "UTSMode": "",
            "UsernsMode": "",
            "ShmSize": 67108864,
            "Runtime": "runc",
            "ConsoleSize": [
                0,
                0
            ],
```

```
            "Isolation": "",
            "CpuShares": 0,
            "Memory": 0,
            "CgroupParent": "",
            "BlkioWeight": 0,
            "BlkioWeightDevice": null,
            "BlkioDeviceReadBps": null,
            "BlkioDeviceWriteBps": null,
            "BlkioDeviceReadIOps": null,
            "BlkioDeviceWriteIOps": null,
            "CpuPeriod": 0,
            "CpuQuota": 0,
            "CpusetCpus": "",
            "CpusetMems": "",
            "Devices": [],
            "DiskQuota": 0,
            "KernelMemory": 0,
            "MemoryReservation": 0,
            "MemorySwap": 0,
            "MemorySwappiness": -1,
            "OomKillDisable": false,
            "PidsLimit": 0,
            "Ulimits": null,
            "CpuCount": 0,
            "CpuPercent": 0,
            "IOMaximumIOps": 0,
            "IOMaximumBandwidth": 0
        },
        "GraphDriver": {
            "Name": "aufs",
            "Data": null
        },
        "Mounts": [
            {
                "Name":
"8917b6dbda7a4948564cef1c0bd3be762fb7e740b161a11d352661bc26cda51a",
                "Source":
"/var/lib/docker/volumes/8917b6dbda7a4948564cef1c0bd3be762fb7e740b161a
11d352661bc26cda51a/_data",
                "Destination": "/var/lib/mysql",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            }
        ],
        "Config": {
            "Hostname": "b7eb67dda521",
            "Domainname": "",
            "User": "",
```

```
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "ExposedPorts": {
                "3306/tcp": {}
            },
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "MYSQL_ROOT_PASSWORD=my-secret-pw",
"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "GOSU_VERSION=1.7",
                "MYSQL_MAJOR=5.7",
                "MYSQL_VERSION=5.7.15-1debian8"
            ],
            "Cmd": [
                "mysqld"
            ],
            "Image": "mysql:latest",
            "Volumes": {
                "/var/lib/mysql": {}
            },
            "WorkingDir": "",
            "Entrypoint": [
                "docker-entrypoint.sh"
            ],
            "OnBuild": null,
            "Labels": {}
        },
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID":
"0cb8bf4fcc4289f3632b63c57eb2d85604abdb5f1d8fa8455a3b5b2a648077ab",
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "Ports": {
                "3306/tcp": null
            },
            "SandboxKey": "/var/run/docker/netns/0cb8bf4fcc42",
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID":
"b36d1403ef312aee22f378e2a4f76935f011e436541a166e38a27aa6138b76c4",
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.2",
```


Alyssa Robinson, lyssanr@yahoo.com

```
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "MacAddress": "02:42:ac:11:00:02",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID":
"599cb4f4bce7afc3b986aefbcb37ebe1d451ce8b6fe4934fe010143b06b485b4",
                    "EndpointID":
"b36d1403ef312aee22f378e2a4f76935f011e436541a166e38a27aa6138b76c4",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:02"
                }
            }
        }
    }
]
```

5.3 Dockerfile with Secrets Specified

When secrets are stored directly in the Dockerfile, they can be viewed by anyone with access to the source code, and the Docker image can be reverse-engineered using tools like dockerfile-from-image (Close, 2015).

```
# Example Dockerfile for imaginary database

FROM ubuntu
MAINTAINER lyssanr@yahoo.com

ENV FAKE_DB_VERSION 1.0.2

#  Install webserver package
RUN apt-get update && apt-get install -y my-fake-db-$FAKE_DB_VERSION

USER fakedb

# Start database and set admin password
RUN    /etc/init.d/myfakedb start &&\
    fakedb --cmd "CREATE USER dbuser WITH PASSWORD 'Password123';" &&\
    createdb

# Open port for myfakedb
EXPOSE 1234
```

Alyssa Robinson, lyssanr@yahoo.com

```
# Add a volume for db data
VOLUME  ["/var/myfakedb", "/etc/fakedb-config"]

# Default container start command
CMD ["/usr/sbin/start_myfakdeb"]
```

Using the docker history command against a docker image built from the Docker example PostgreSQL, which contains several passwords:

```
[ec2-user@ip-192-168-0-213 ~]$ docker images
REPOSITORY                                  TAG                 IMAGE ID
CREATED             SIZE
eg_postgresql                               latest
c8c885dfb805        11 minutes ago      393.9 MB
ubuntu                                      latest
f753707788c5        2 weeks ago         127.1 MB
centurylink/dockerfile-from-image           latest
970eaf375dfd        10 months ago       19.16 MB

ubuntu@ip-192-168-0-235:~$ docker history --no-trunc eg_postgresql
|awk '{$1=$2=$3=$4=$5=""; print $0}'
WARNING: Error loading config file:/home/ubuntu/.dockercfg - read
/home/ubuntu/.dockercfg: is a directory
     COMMENT
     -c #(nop) CMD ["/usr/lib/postgresql/9.3/bin/postgres" "-D"
"/var/lib/postgresql/9.3/main" "-c"
"config_file=/etc/postgresql/9.3/main/postgresql.conf"] 0 B
     -c #(nop) VOLUME [/etc/postgresql /var/log/postgresql
/var/lib/postgresql] 0 B
     -c #(nop) EXPOSE 5432/tcp 0 B
     -c echo "listen_addresses='*'" >>
/etc/postgresql/9.3/main/postgresql.conf 20.32 kB
     -c echo "host all all 0.0.0.0/0 md5" >>
/etc/postgresql/9.3/main/pg_hba.conf 4.681 kB
     -c /etc/init.d/postgresql start && psql --command "CREATE USER
docker WITH SUPERUSER PASSWORD 'docker';" && createdb -O docker docker
23.43 MB
     -c #(nop) USER [postgres] 0 B
     -c apt-get update && apt-get install -y python-software-
properties software-properties-common postgresql-9.3 postgresql-
client-9.3 postgresql-contrib-9.3 243.3 MB
     -c echo "deb http://apt.postgresql.org/pub/repos/apt/ precise-
pgdg main" > /etc/apt/sources.list.d/pgdg.list 63 B
     -c apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 -
-recv-keys B97B0AFCAA1A47F044F244A07FCC7D46ACCC4CF8 27.72 kB
     -c #(nop) MAINTAINER SvenDowideit@docker.com 0 B
     -c #(nop) CMD ["/bin/bash"] 0 B
     -c mkdir -p /run/systemd && echo 'docker' >
/run/systemd/container 7 B
     -c sed -i 's/^#\s*\(deb.*universe\)$/\1/g' /etc/apt/sources.list
```

Alyssa Robinson, lyssanr@yahoo.com

```
1.895 kB
     -c rm -rf /var/lib/apt/lists/* 0 B
     -c set -xe && echo '#!/bin/sh' > /usr/sbin/policy-rc.d && echo
'exit 101' >> /usr/sbin/policy-rc.d && chmod +x /usr/sbin/policy-rc.d
&& dpkg-divert --local --rename --add /sbin/initctl && cp -a
/usr/sbin/policy-rc.d /sbin/initctl && sed -i 's/^exit.*/exit 0/'
/sbin/initctl && echo 'force-unsafe-io' > /etc/dpkg/dpkg.cfg.d/docker-
apt-speedup && echo 'DPkg::Post-Invoke { "rm -f
/var/cache/apt/archives/*.deb /var/cache/apt/archives/partial/*.deb
/var/cache/apt/*.bin || true"; };' > /etc/apt/apt.conf.d/docker-clean
&& echo 'APT::Update::Post-Invoke { "rm -f
/var/cache/apt/archives/*.deb /var/cache/apt/archives/partial/*.deb
/var/cache/apt/*.bin || true"; };' >> /etc/apt/apt.conf.d/docker-clean
&& echo 'Dir::Cache::pkgcache ""; Dir::Cache::srcpkgcache "";' >>
/etc/apt/apt.conf.d/docker-clean && echo 'Acquire::Languages "none";'
> /etc/apt/apt.conf.d/docker-no-languages && echo
'Acquire::GzipIndexes "true"; Acquire::CompressionTypes::Order::
"gz";' > /etc/apt/apt.conf.d/docker-gzip-indexes && echo
'Apt::AutoRemove::SuggestsImportant "false";' >
/etc/apt/apt.conf.d/docker-autoremove-suggests 745 B
     -c #(nop) ADD
file:b1cd0e54ba28cb1d6db4b93f98d5e02f5e2bcd96c55cb91731ba499861001e30
in / 127.2 MB
```

6. Base Image Security

6.1 Checklist:

- ☐ Specify package versions and hash in FROM tag
- ☐ Understand the contents of running images
- ☐ Scan images regularly for vulnerabilities
- ☐ Update base image regularly and re-deploy containers

a. Example Dockerfile with specification of package versions in FROM tag:

```
FROM
ubuntu:16.04@sha256:3235a49037919e99696d97df8d8a230717272d848ee4d
dadbca8d54f97ee30cb


ENV MY_APP_VERSION=0.734.
RUN apt-get update && \
    DEBIAN_FRONTEND=noninteractive apt-get install -y my-
app=${MY_APP_VERSION}* && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

WORKDIR /my_app_dir
```

Alyssa Robinson, lyssanr@yahoo.com

b. Image vulnerability scanning using CoreOS Clair:

```
ubuntu@ip-192-168-0-43:~$ docker images
REPOSITORY                                TAG                 IMAGE ID
CREATED             SIZE
oscap4docker                              latest
d5dbedfe803f        3 hours ago         406.3 MB
containercompliance_oscap4docker-test     latest
3581f036e997        3 hours ago         256.1 MB
<none>                                    <none>
d30ce74503e7        3 hours ago         406.3 MB
postgres                                  latest
654b61cc82aa        11 days ago         264.7 MB
ubuntu                                    latest
c73a085dc378        2 weeks ago         127.1 MB
nginx                                     latest
ba6bed934df2        2 weeks ago         181.4 MB
mysql                                     latest
18f13d72f7f0        2 weeks ago         383.4 MB
centos                                    7
980e0e4c79ec        4 weeks ago         196.8 MB
quay.io/coreos/clair                      latest
a76bf177c731        12 weeks ago        836.6 MB
hello-world                               latest
c54a2cc56cbb        3 months ago        1.848 kB
dduportal/bats                            0.4.0
e055fece8340        8 months ago        149.9 MB
dduportal/oscap4docker                    1.0.0
5f272e73d4e3        16 months ago       366.6 MB

ubuntu@ip-192-168-0-43:~$ analyze-local-images 654b61cc82aa
2016-10-11 01:47:19.316667 I | Saving 654b61cc82aa to local disk (this
may take some time)
2016-10-11 01:47:25.220053 I | Retrieving image history
2016-10-11 01:47:25.220199 I | Analyzing 11 layers...
2016-10-11 01:47:25.220207 I | Analyzing
7f1d5a7e7c722d89fb14528ffde7ef6002c737cd4c5e694306e67934f8baee3e
2016-10-11 01:47:25.222376 I | Analyzing
d12a8ea4eea45b28283f0edf8058b94816708a6d110753b3ca2d3ee984f55665
2016-10-11 01:47:25.223978 I | Analyzing
0c3d09bc94687bfc3b55b93d1046d82acfaac6ec0609a6b1e622ee4c7ecc396d
2016-10-11 01:47:25.225271 I | Analyzing
37859a7bd1ae7e4a5b55a1fecd46f12590b8cc8ca7e902d00d50b6f61dfa76d3
2016-10-11 01:47:25.226739 I | Analyzing
2890c2e45866015721938d3e02dc74d7457656fb7626c7f224bb20545abddd97
2016-10-11 01:47:25.228183 I | Analyzing
2f5f35ab6e98d31fdf2916e0fa10500bd27cc15266064dc5edd67bc756e834c3
2016-10-11 01:47:25.229650 I | Analyzing
1424eee3083d5defbc52d6566d351c40c90ef92fbb697cf443c08c64dd615ff0
2016-10-11 01:47:25.230918 I | Analyzing
```

```
5520766bc8a3577e2faedce8af3c6ccefa5af1b78436b8d416f7784731fe6ded
2016-10-11 01:47:25.232418 I | Analyzing
0b6bb94658de3448349db8edfe5fedb2f5fc4668daa88af4147032f395d67f19
2016-10-11 01:47:25.233852 I | Analyzing
19c1a9a264b62abda7a801086bab5558ab98d17e73c89bc671a670efbd12d2e2
2016-10-11 01:47:25.235211 I | Analyzing
fc08ba35d0edff25393a24067d20310dbbf6997f45b56e993e1d98589cf9d24c
2016-10-11 01:47:25.236456 I | Retrieving image's vulnerabilities
Clair report for image 654b61cc82aa (2016-10-11 01:47:25.243500896
+0000 UTC)
Success! No vulnerabilities were detected in your image

ENTRYPOINT ["/bin/myapp"]
```

7. Limit container resources

7.1 Checklist:

- ☐ Mount filesystems read-only (prevent writing malicious code)
- ☐ Place limits on system resources (limit denial of service)
- ☐ Limit kernel calls (limit container breakout) and linux capabilities
- ☐ Restrict network access (prevent attack pivot, data egress, -icc=false)

7.2 Docker_bench run on default mysql container:

```
ubuntu@ip-192-168-0-235:~$ docker run --name some-mysql -u mysql -e
MYSQL_ROOT_PASSWORD=adhagohrgh9876 -d mysql:latest
5601ba7ab19e9e151e63867a1194ef6877cdeb527fd22331358266a335ccea0e
ubuntu@ip-192-168-0-235:~$ docker ps
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS              PORTS               NAMES
5601ba7ab19e        mysql:latest        "docker-entrypoint.sh"   3
seconds ago       Up 2 seconds        3306/tcp            some-mysql


ubuntu@ip-192-168-0-235:~$ docker run -it --net host --pid host --cap-
add audit_control     -v /var/lib:/var/lib     -v
/var/run/docker.sock:/var/run/docker.sock     -v
/usr/lib/systemd:/usr/lib/systemd     -v /etc:/etc --label
docker_bench_security     docker/docker-bench-security
# ------------------------------------------------------------------
----------
# Docker Bench for Security v1.1.0
#
# Docker, Inc. (c) 2015-
#
# Checks for dozens of common best-practices around deploying Docker
containers in production.
# Inspired by the CIS Docker 1.11 Benchmark:
# https://benchmarks.cisecurity.org/downloads/show-
single/index.cfm?file=docker16.110
```

Alyssa Robinson, lyssanr@yahoo.com

```
# ------------------------------------------------------------
---------

[INFO] 5  - Container Runtime
[WARN] 5.1  - Verify AppArmor Profile, if applicable
[WARN]      * No AppArmorProfile Found: some-mysql
[WARN] 5.2  - Verify SELinux security options, if applicable
[WARN]      * No SecurityOptions Found: some-mysql
[PASS] 5.3  - Restrict Linux Kernel Capabilities within containers
[PASS] 5.4  - Do not use privileged containers
[PASS] 5.5  - Do not mount sensitive host system directories on
containers
[PASS] 5.6  - Do not run ssh within containers
[PASS] 5.7  - Do not map privileged ports within containers
[PASS] 5.9 - Do not share the host's network namespace
[WARN] 5.10 - Limit memory usage for container
[WARN]      * Container running without memory restrictions: some-
mysql
[WARN] 5.11 - Set container CPU priority appropriately
[WARN]      * Container running without CPU restrictions: some-mysql
[WARN] 5.12 - Mount container's root filesystem as read only
[WARN]      * Container running with root FS mounted R/W: some-mysql
[PASS] 5.13 - Bind incoming container traffic to a specific host
interface
[WARN] 5.14 - Set the 'on-failure' container restart policy to 5
[WARN]      * MaximumRetryCount is not set to 5: some-mysql
[PASS] 5.15 - Do not share the host's process namespace
[PASS] 5.16 - Do not share the host's IPC namespace
[PASS] 5.17 - Do not directly expose host devices to containers
[INFO] 5.18 - Override default ulimit at runtime only if needed
[INFO]      * Container no default ulimit override: some-mysql
[PASS] 5.19 - Do not set mount propagation mode to shared
[PASS] 5.20 - Do not share the host's UTS namespace
[PASS] 5.21 - Do not disable default seccomp profile
[PASS] 5.24 - Confirm cgroup usage
[WARN] 5.25 - Restrict container from acquiring additional privileges
[WARN]      * Privileges not restricted: some-mysql
```

7.3 Docker bench with limited container resources:

```
ubuntu@ip-192-168-0-235:~$ docker run -m 256m -c 512 --name some-nginx
--read-only -v /var/cache/nginx -v /var/run -v /dev/log:/dev/log --
cap-drop SETUID --cap-drop SETGID --security-opt="no-new-privileges" -
-restart on-failure:5 -d nginx
ubuntu@ip-192-168-0-235:~$ docker run -it --net host --pid host --cap-
add audit_control     -v /var/lib:/var/lib     -v
/var/run/docker.sock:/var/run/docker.sock     -v
/usr/lib/systemd:/usr/lib/systemd     -v /etc:/etc --label
docker_bench_security     docker/docker-bench-security
# ------------------------------------------------------------
---------
```

Alyssa Robinson, lyssanr@yahoo.com

```
# Docker Bench for Security v1.1.0
#
# Docker, Inc. (c) 2015-
#
# Checks for dozens of common best-practices around deploying Docker
containers in production.
# Inspired by the CIS Docker 1.11 Benchmark:
# https://benchmarks.cisecurity.org/downloads/show-
single/index.cfm?file=docker16.110
# ------------------------------------------------------------------
---------

[INFO] 5  - Container Runtime
[WARN] 5.1  - Verify AppArmor Profile, if applicable
[WARN]      * No AppArmorProfile Found: some-nginx
[PASS] 5.2  - Verify SELinux security options, if applicable
[PASS] 5.3  - Restrict Linux Kernel Capabilities within containers
[PASS] 5.4  - Do not use privileged containers
[PASS] 5.5  - Do not mount sensitive host system directories on
containers
[PASS] 5.6  - Do not run ssh within containers
[PASS] 5.7  - Do not map privileged ports within containers
[PASS] 5.9 - Do not share the host's network namespace
[PASS] 5.10 - Limit memory usage for container
[PASS] 5.11 - Set container CPU priority appropriately
[PASS] 5.12 - Mount container's root filesystem as read only
[PASS] 5.13 - Bind incoming container traffic to a specific host
interface
[PASS] 5.14 - Set the 'on-failure' container restart policy to 5
[PASS] 5.15 - Do not share the host's process namespace
[PASS] 5.16 - Do not share the host's IPC namespace
[PASS] 5.17 - Do not directly expose host devices to containers
[INFO] 5.18 - Override default ulimit at runtime only if needed
[INFO]      * Container no default ulimit override: some-nginx
[PASS] 5.19 - Do not set mount propagation mode to shared
[PASS] 5.20 - Do not share the host's UTS namespace
[PASS] 5.21 - Do not disable default seccomp profile
[PASS] 5.24 - Confirm cgroup usage
[PASS] 5.25 - Restrict container from acquiring additional privileges
```