Top of Form

Bottom of Form

**[Linux Audit](http://linux-audit.com/" \o "Linux Audit)**

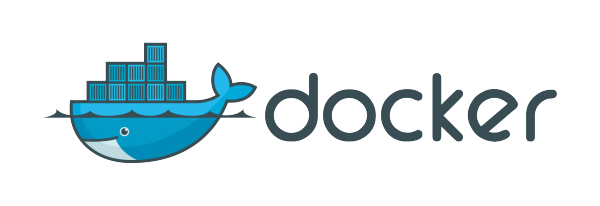
**Linux security: Auditing, Hardening and Compliance**

* [**Home**](http://linux-audit.com/)
* [**Lynis**](http://linux-audit.com/lynis/)
* [**Contact**](http://linux-audit.com/contact/)
* [**About Linux Audit**](http://linux-audit.com/about/)

[January 22, 2015](http://linux-audit.com/docker-security-best-practices-for-your-vessel-and-containers/) [Michael Boelen](http://linux-audit.com/author/mboelen/)[Automation](http://linux-audit.com/category/automation/), [Containers](http://linux-audit.com/category/containers/), [Docker](http://linux-audit.com/category/docker/), [Virtualization](http://linux-audit.com/category/virtualization/) [Leave a comment](http://linux-audit.com/docker-security-best-practices-for-your-vessel-and-containers/#respond)

**Docker Security: Best Practices for your Vessel and Containers**

**Docker Security**

*Everything you need to know about Docker security.*

**Introduction into Docker**

Operating systems like [CoreOS](https://coreos.com/) use Docker to run applications on top of their lightweight platform. Docker in its turn, provides utilities around technologies like Linux container technology (e.g. LXC, systemd-nspawn, libvirt). Previously Docker could be described as the “automated LXC”, now it’s actually even more powerful. What it definitely is, is simplifying and enhancing the possibilities of container technology.

**Continuous delivery**

Rolling out containers is quick and very easy. It helps companies to improve development work flows. Developers can perform testing and deploy their applications much easier than before. Additionally, the use of Docker enhances the process from development, up to running software in production. This is achieved by using smaller units, which are easier to create, monitor and secure.

**Supporting multiple technologies**

Docker uses the possibilities of the hypervisor management tooling like libvirt and systemd-nspawn. It has an ongoing development and supports more and more features, to simplify the management of containers. With each of the component getting more stable, the base of Docker reached a level of stability, use it in production.

**Enhancing security**

With the right measures, Docker will also enhance security. For example due to running (smaller) individual units, controlling them is easier. One benefit of small containers is providing application owners and administrators a better insight what software, protocols and network flows are needed for individual services.

IT architects and security professionals will definitely benefit from container technology as well. Architects gain fine grained building blocks, to define new services and enhance exiting ones. Security professionals benefit from a better segmentation, and minimizing the permissions needed in each individual container.

**Docker Security and Risks**

Software packages can solve existing security threats, or actually introduce new risks. This is also the case when using Docker. While it can help in reducing risks by using compartmentalization, the implementation might have its flaws.

**Risks**

One common threat of new services is a low(er) stability, which forms a risk to the availability of a service. Another one is information disclosure, as the service might be lacking appropriate controls. Usually new technologies have a need of adding new features quickly, which might result in sloppy programming. Often this will result in software vulnerabilities, including ones which are security related.

Unfortunately Docker already had its share of security vulnerabilities, but they took a more active stance to [improve the security](http://blog.docker.com/2014/12/advancing-docker-security-docker-1-4-0-and-1-3-3-releases/)of their products.

**Methods and best practices**

To reduce the risks when using fairly new technologies, we will have a look at the methods available to Docker. In particular, we look how Docker can increase security. After that, we provide some best practices when dealing with Docker containers.

**Maturity level**  
Container management is a fairly new technology, which leaves many professionals with a knowledge gap in this area. Additionally, not many people are capable (yet) of making a proper security assessment. For example, auditors might not be able to ask the right questions regarding the implementation of containers.

This mature level risk also includes lacking technical auditing of containers, or difficulties in maintaining proper and up-to-date documentation. After all, the flexibility containers provide also means containers can run more easily on a different system, at any given time. This might need another level of documentation, to reflect the possibilities of each individual unit.

**Containers do not (fully) contain**

While containers are used to compartmentalize and limit resources, they actually don’t fully contain. For example a process running in the container with UID 1000, will have the privileges of UID 1000 on the underlying host as well. Along these lines, a process running as root (UID 0) in a container has root-level privileges on the underlying host when interacting with the kernel.

This risk will be soon be mitigated, when user namespaces are fully being implemented. The first step is already made in the form of subordinate user IDs (subuid). This helps mapping existing user IDs on the host system, into different user IDs within each container.

**Security Benefits of Docker**

**Segregation of applications**

Normally applications all run on the same host system. By using container technology we can segregate them, making it easier to determining traffic flows.

**Flexible attitude**

With containers being smaller individual units, they become dynamic, or flexible. The work flow to maintain them is more flexible as well. Great for security patching, testing and releasing the updated containers into production.

**Focus on automation**

Docker has a clear focus on automation. They have supporting products like [Docker Machine](https://github.com/docker/machine), [Swarm](https://github.com/docker/swarm) (clustering) and Compose, to simplify management of many containers.

**Limiting information disclosure**

Containers can limited resources assigned. This helps us in limiting the amount of information available to the system (and an evil attacker). Each containers gets the following components:

* Network stack
* Process space
* File system instances

Limiting resources are achieved by using namespaces. Namespaces are like a “view”, which only shows a subset of all resources on the system. This provides a form of isolation: processes running within a container cannot see, or affect, processes in other containers, or the host system itself.

**Protection Methods**

By using Docker properly, some of its defenses can be used. Unfortunately the tooling does not *actively* help yet to leverage all possibilities. You are the one to properly configure, use and update Docker. Our hope is that Docker in the future will be more strict with some options, or at least advise the user to some extent.

**Limited capabilities**

Linux has support for “capabilities”, which can be seen as roles. A role could be opening a network socket, to craft a packet and put it onto the wire. Normally these kind of roles are only available to the root user. By splitting them in the form of capabilities, they can also be assigned to individual processes as well.  This way a piece of software can still open up a socket (with “root permissions”), while not being able to load a new kernel driver. For more details about capabilities, see our previous blog post [Linux Capabilities 101](http://linux-audit.com/linux-capabilities-101/).

Containers will run with a limited capability set. So even if someone breaks into the container, the host system is to some extent protected.

**Examples:**

* Mounting operations
* Access to raw sockets (prevent opening privileged ports, spoofing)
* Some file system operations (mkdev, chown, chattrs)
* Loading kernel modules

The configuration and usage of capabilities will be covered later. For now it is good to know that Docker by default drops a list of capabilities:

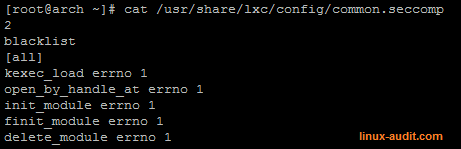
* CAP\_AUDIT\_WRITE = Audit log write access
* CAP\_AUDIT\_CONTROL = Configure Linux Audit subsystem
* CAP\_MAC\_OVERRIDE = Override kernel MAC policy
* CAP\_MAC\_ADMIN = Configure kernel MAC policy
* CAP\_NET\_ADMIN = Configure networking
* CAP\_SETPCAP = Process capabilities
* CAP\_SYS\_MODULE = Insert and remove kernel modules
* CAP\_SYS\_NICE = Priority of processes
* CAP\_SYS\_PACCT = Process accounting
* CAP\_SYS\_RAWIO = Modify kernel memory
* CAP\_SYS\_RESOURCE = Resource Limits
* CAP\_SYS\_TIME = System clock alteration
* CAP\_SYS\_TTY\_CONFIG = Configure tty devices
* CAP\_SYSLOG = Kernel syslogging (printk)
* CAP\_SYS\_ADMIN = All others

**Usage of seccomp**

Secure Computing, or seccomp, helps with the creation of sandboxes. It does so by defining what system calls should be blocked. The latest version of seccomp provides this syscall filtering by using the Berkeley Packet Filter (BPF), previously used for filtering network traffic.

Containers currently have the following syscalls disabled (since LXC 1.0.5):

* kexec\_load
* open\_by\_handle\_at
* init\_module
* finit\_module
* delete\_module

[](http://linux-audit.com/wp-content/uploads/2015/01/linux-audit-linux-container-seccomp-options.png)

System calls blocked by default in Linux containers

When any of the blocked syscalls is made, the kernel will send a SIGKILL signal to stop the related process.

**Digital Signature Verification**

Starting with Docker version 1.3.0 all images are verified after downloading. This is a great step in enhancing the level of trust for downloads. This is why we have done this for our auditing tool Lynis as well. Like our website, Docker is also providing their website HTTPS-only. Another level of trust and ensuring you are are the right place, downloading the tools from Docker.

**Current issues with Docker**

**Root permissions**

Right now there is a small issue left with Docker, which is the requirement of running the daemon with root permissions. Docker is aware of it and plans to define well-audited sub-processes, which do not longer require root permissions. Additionally, each sub-process will run with a very limited scope, increasing the security level of each component and enhancing stability.

**Lack of full User namespace implementation**

Currently there is still no full user namespace implementation. Something which is out of the control of Docker. When the LXC userland tools are evolved and include the support, Docker can leverage the possibilities. The first actions like user mapping is available, so full support is expected soon.

**User 0 in container = User 0 on host**  
One of the risks due to the missing User namespaces, is that the mapping of users from the host to the containers is still an one-to-one mapping. Previously user 0 in the container was equal to 0 on the host. In other words, if you container is compromised, it doesn’t take much to compromise the full host. Fortunately this is work in progress. LXC already supports a mapping option, to map user ID 0 in the container to another (high) ID on the host.

**Default allow all**

By default all IP traffic is allowed between containers. This means they can ping each other, but also send other forms of traffic. It would have been better if Docker applied a “deny all by default” principle. This forces the maintainer of the container to think about what kind of traffic is needed between individual containers.

Fortunately traffic can be filtered and is absolute advised for systems in production.

**Best Practices**

With all these risks and possibilities, lets extract some of the lessons. These best practices help you create more safe services and enhance the security of existing containers.

**Do not run software as root**

This tip might sound to simple, but still many developers run their software as the root user. Containers still can not contain properly, which might result in a full host compromise if a container is compromised. Therefore, run your software packages like you would run them on a normal host.

**Use Docker in combination with AppArmor/SELinux/TOMOYO**

Ubuntu comes with ready-to-use AppArmor templates for LXC. It is always a good thing to know what your software does. This includes knowing what paths and permissions your software components need to function properly. With this information each piece can be restricted to the bare minimum needed, preventing permission escalation and unauthorized information disclosure (or worse).

To achieve the right policies, make sure to monitoring your applications from the start, including the related framework you are using. Each of them provides the means to monitor, so use them.

Within the container configuration the related AppArmor profile can be defined with **lxc.aa\_profile**.

**Use seccomp to limit syscalls**

Support for seccomp is available to (at least) CentOS, Debian, Fedora, Gentoo, Oracle, Plamo and Ubuntu. You can use seccomp by altering the container configuration and define the seccomp rule set to be used:

lxc.seccomp = /usr/share/lxc/config/common.seccomp

For Docker this functionality can be activated by using the **–lxc-conf** parameter to **docker run**.

LXC configuration option: lxc.seccomp

**Limit traffic with iptables**

By default all containers use the docker0 interface as a bridge. Like on a normal host you can limit traffic, to block unauthorized traffic streams.

For full details we suggest to read the [advanced networking](https://docs.docker.com/articles/networking/) blog post at Docker.

**GRSEC and PaX**

When possible, use a hardened Linux kernel, with kernel patches. [Grsecurity](https://grsecurity.net/) and PaX are two examples which help in hardening the host system.

**Using user mappings**

To counter the issue that user 0 in a particular container equals root on the host system, LXC allows you to remap user and group IDs. The configuration file entry would look something like this:

lxc.id\_map = u 0 100000 65536  
lxc.id\_map = g 0 100000 65536

This maps the first 65536 user and group IDs within the container to 100000-165536 on the host. The related files on the host are /etc/subuid and /etc/subgid. This mapping technique is named subordinate IDs, hence the “sub” prefixes.

For Docker this means adding it as a **-lxc-conf** parameter to docker run:

docker run -lxc-conf=”lxc.id\_map = u 0 100000 65536″ -lxc-conf=”lxc.id\_map = g 0 100000 65536″

**Use –cap-drop and –cap-add**

(since Docker 1.2.0)

With the earlier described Linux capabilities we can tell Docker what specific roles should be given to a container. Actually, we can use both the add and drop option together. By first allowing all, then dropping some capabilities, we can limit the capabilities. The short version is just dropping permissions. The alternative is just adding the related capabilities, which more resembles the “deny all” principle.

docker run –cap-drop=CHOWN  
docker run –cap-add=ALL –cap-drop=MKNOD

A (fairly) safe list of capabilities to drop are:

* audit\_control
* audit\_write
* mac\_admin
* mac\_override
* mknod
* setfcap setpcap
* sys\_admin
* sys\_boot
* sys\_module
* sys\_nice
* sys\_pacct
* sys\_rawio
* sys\_resource
* sys\_time
* sys\_tty\_config

See the capabilities(7) man page for all details about these capabilities.

LXC configuration options: lxc.cap.drop and lxc.cap.keep

**Do not run SSH in containers**

Use “docker exec -it mycontainer bash” instead to manage your containers.

**Do not run –privileged on containers**

(since Docker 1.3.0)

For containers which already have SELinux/AppArmor support, use **–security-opt** instead. This gives it the appropriate security profile, instead of giving away too much permissions within the container.

docker run –security-opt label:type:svirt\_apache -i -t centos \ bash

Related options for SELinux:

* –security-opt=”label:user:USER” (set label user)
* –security-opt=”label:role:ROLE” (set label role)
* –security-opt=”label:type:TYPE” (set label type)
* –security-opt=”label:level:LEVEL” (set label level)
* –security-opt=”label:disable” (disable label confinement completely)

Options for AppArmor:

* –secutity-opt=”apparmor:PROFILE” (set AppArmor profile)

For more options have a look at the [Docker Run Reference](https://docs.docker.com/reference/run/).

**Upgrade your Docker version on a regular basis**

Most software packages have bugs, small programming errors. With Docker also being under heavy development, bugs are solved and new features added. Regular updating and making Docker part of your software patch management process, is advised.

**Secure Docker client connections**

Set the **DOCKER\_HOST** and **DOCKER\_TLS\_VERIFY** variable, to use TLS for connecting to Docker instances. See<https://docs.docker.com/articles/https/> for detailed instructions on setting this up.

**Vulnerabilities in Docker**

Over the years vulnerabilities will show up. Some of the Docker security messages are collected for archival purposes:

Docker 1.3.3 fixes: <https://groups.google.com/forum/#!msg/docker-user/nFAz-B-n4Bw/0wr3wvLsnUwJ>  
Docker 1.3.2 fixes: <https://groups.google.com/forum/#!topic/docker-user/IrjXTHA6jJc>  
Docker 1.3.1 fixes:  CVE-2014-5277 and CVE-2014-3566 <https://groups.google.com/forum/#!topic/docker-user/oYm0i3xShJU>

**History**

July 2008 – Kernel namespaces introduced

March 2013 – Initial release of Docker

October 2014 – Docker release 1.3

January 2015 – First release of this article

**Citing Sources**

Used references and sources, include the following websites:

* <http://blog.docker.com/2013/08/containers-docker-how-secure-are-they/> – Blog of Docker
* <http://blog.zeltser.com/post/104976675349/security-risks-and-benefits-of-docker-application> – Blog of Lenny Zeltser
* <https://linuxcontainers.org/lxc/news/> – Useful changelogs for LXC
* <https://docs.docker.com/reference/run/> – Docker Run Reference

**Changes**

Found something outdated in this article? Add it to the comments. This article is kept up-to-date on a regular basis, together with the developments of Docker itself.

* [**capabilities**](http://linux-audit.com/tag/capabilities/)
* [**containers**](http://linux-audit.com/tag/containers/)
* [**docker**](http://linux-audit.com/tag/docker/)
* [**docker security**](http://linux-audit.com/tag/docker-security/)
* [**linux**](http://linux-audit.com/tag/linux/)
* [**seccomp**](http://linux-audit.com/tag/seccomp/)
* [**security**](http://linux-audit.com/tag/security/)

[](https://cisofy.com/lynis-enterprise/)

Liked the article? If you like to automate things as much as we do, you may like our tools. Besides sharing information on this Linux security blog, we help companies with security automation, and testing their security defenses.

**Lynis and Lynis Enterprise**

Lynis is our open source security scanner. Tailored to those who need to secure Linux and UNIX systems. Focus is on automated security audits and system hardening. Simple to use, and already being used by many system administrators, auditors, and security professionals.

Doing management for 10+ systems? Lynis Enterprise will simplify your work: central management, reporting, dashboards, hardening snippets, and more. Use cases include: File Integrity Monitoring, Compliance, Continuous Auditing, Intrusion Detection, System Hardening and Vulnerability Management.

[**Show me!**](https://cisofy.com/lynis-enterprise/)

**Leave a Reply**

Top of Form

Your email address will not be published. Required fields are marked \*

Comment

Name \*

Email \*

Website



Bottom of Form

**About Linux Audit**

This blog is part of our mission: help individuals and companies, to secure their systems, and comply with regulations. We simply love Linux security and system auditing.

Besides the blog, we have our security auditing tool Lynis. Open source, GPL, and free to use.

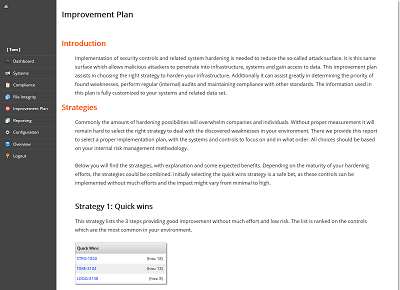
[**Download Lynis (Free)**](https://cisofy.com/downloads/)

For those with enterprise needs, or want to audit multiple systems, there is an [Enterprise](https://cisofy.com/lynis-enterprise/)version.

*"One security solution to audit, harden, and secure your Linux/UNIX systems."*

**Benefits:**

* Perform audits within a few minutes
* Central management
* Powerful reporting
* Compliance checks (e.g. PCI DSS)
* Additional plugins and more tests



[**Learn More**](https://cisofy.com/lynis-enterprise/)

Enjoy the articles!

**Recent Posts**

* [Best Open Source Security Tools of 2015 (and 2016)](http://linux-audit.com/best-open-source-security-tools-of-2015-2016/)
* [Delete a HSTS Key Pin in Chrome](http://linux-audit.com/delete-a-hsts-key-pin-in-chrome/)
* [Automatic Security Updates with DNF](http://linux-audit.com/automatic-security-updates-with-dnf/)
* [Dealing with Linux Malware, Insights by the Author of rkhunter](http://linux-audit.com/dealing-with-linux-malware-insights-by-the-author-of-rkhunter/)
* [File Integrity of Password Files](http://linux-audit.com/file-integrity-of-password-files/)
* [Deleting Outdated HPKP Key Pins in Firefox](http://linux-audit.com/deleting-outdated-hpkp-key-pins-in-firefox/)
* [Showing Available Security Updates with DNF](http://linux-audit.com/showing-available-security-updates-with-dnf/)
* [File permissions of the /etc/shadow password file](http://linux-audit.com/file-permissions-of-the-etc-shadow-password-file/)
* [Differences Between iptables and nftables Explained](http://linux-audit.com/differences-between-iptables-and-nftables-explained/)
* [Linux Security Guide for Hardening IPv6](http://linux-audit.com/linux-security-guide-for-hardening-ipv6/)

**Receive Updates**

Outsmart your colleagues and receive the latest updates.

Top of Form

Email\*   




Bottom of Form

Or follow on the social media channels:

**Contact**

This blog is part of our mission to share valuable tips about Linux security. We are reachable via [@linuxaudit](https://twitter.com/linuxaudit)

**Company Details**

CISOfy  
De Klok 28,  
5251 DN, Vlijmen, The Netherlands  
+31202260055

Website: [cisofy.com](https://cisofy.com/)

*© 2013-2016 Linux Audit - By the security experts behind*[*Lynis Enterprise*](https://cisofy.com/lynis-enterprise/)     [Privacy Policy](http://linux-audit.com/privacy-policy/) - [About](http://linux-audit.com/about/)