Docker Security

Mika Vatanen

13.6.2017

About me

- Mika Vatanen, Solution Architect @ Digia
- 18 years at the industry, 6 months at Digia
- Established ii2 – a Finnish MySpace, top-5 most used web service in Finland between 2006-2008
- Wide interest in different new technologies. Always very keenly interested on IT security

Today’s speak

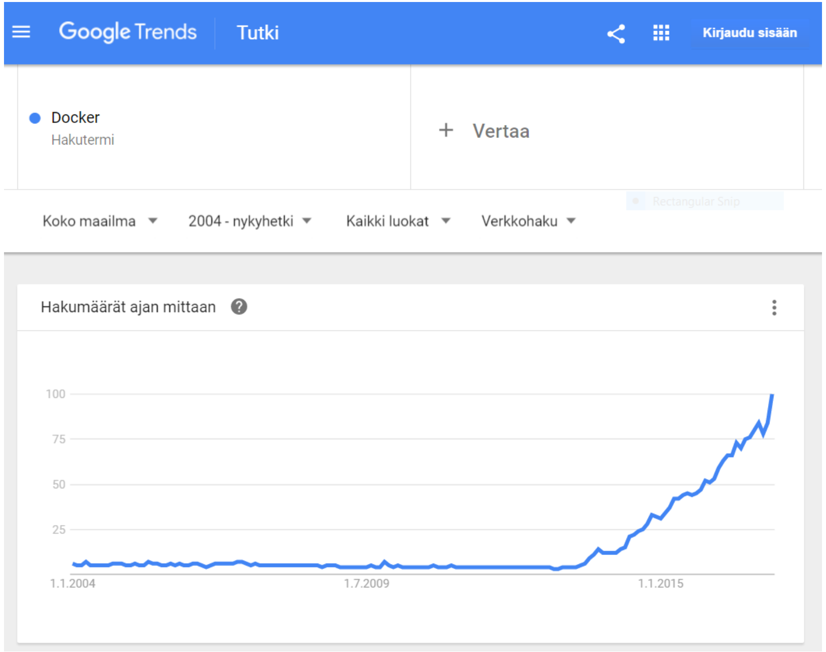
- How does Docker change security landscape?
- What attack vectors are there on Docker. How does Docker handle security?
- How to increase Docker security

Why is Docker security important?

- Docker is currently experiencing very high adoption rate

- Many people are deploying on Docker without considering the security landscape

- Main reason why companies are hesitating on switching to Docker (Forrester, 2015)



More secure, less secure?

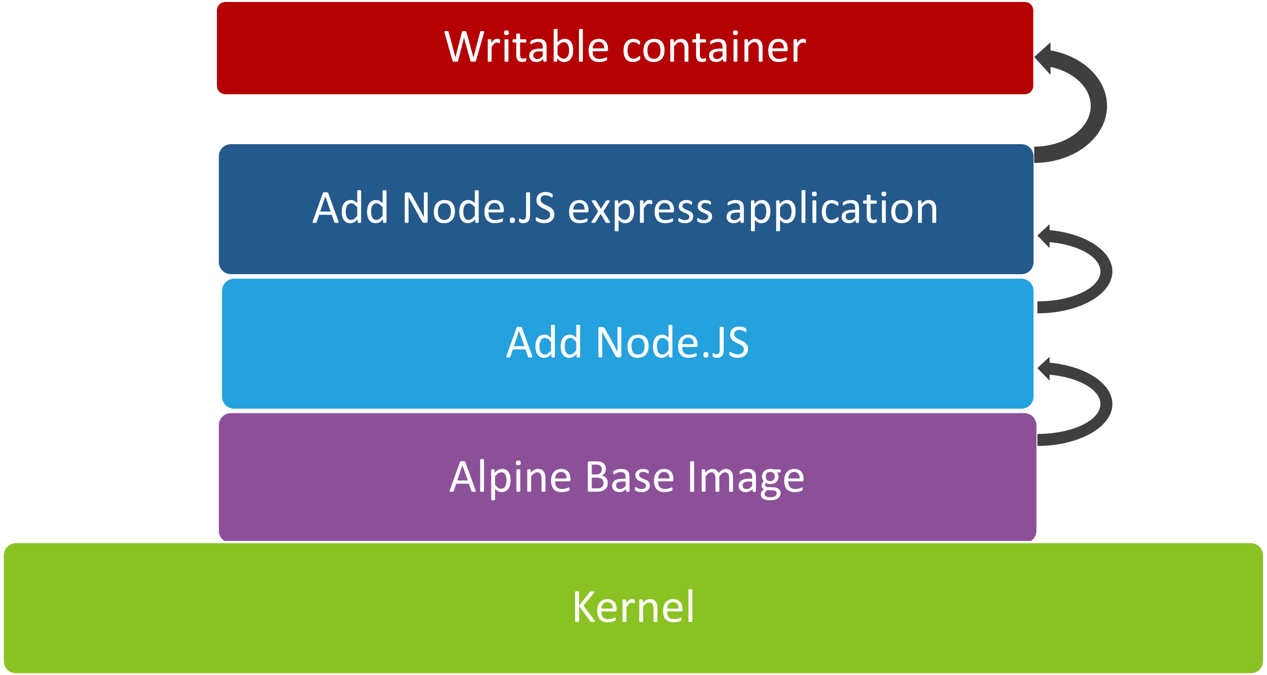
- *“Gartner asserts that applications deployed in containers are more secure than applications deployed on the bare OS […] as long as a kernel privilege escalation vulnerability does not exist on the host OS”*

  (Joerg Fritsch, Research Director, Gartner, 2016)

But…

- *no automated security updates*
- *shift of security from ops to software developers*
- *higher risk if multiple applications are run in shared servers*

Docker Structure



digia

Docker hub image vulnerabilities

- Docker Hub images contain ~180 vulnerabilities on average. Many images have not been updated for hundreds of days
- A security vulnerability introduced at lower layers is propagated into all dependent layers
- Source: A Study of Security Vulnerabilities on Docker Hub, Shu et al. 2017

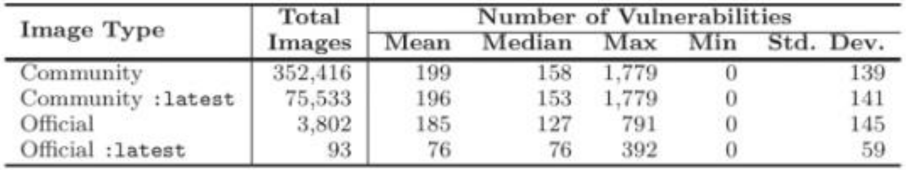
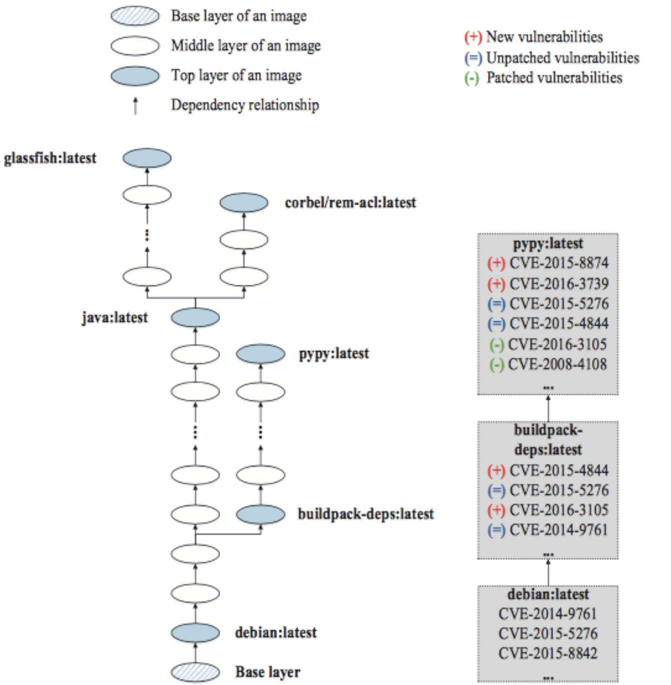


Image Type	Total Images	Number of Vulnerabilities				
		Mean	Median	Max	Min	Std. Dev.
Community	352,416	199	158	1,779	0	139
Community :latest	75,533	196	153	1,779	0	141
Official	3,802	185	127	791	0	145
Official :latest	93	76	76	392	0	59

Number of Vulnerabilities per image

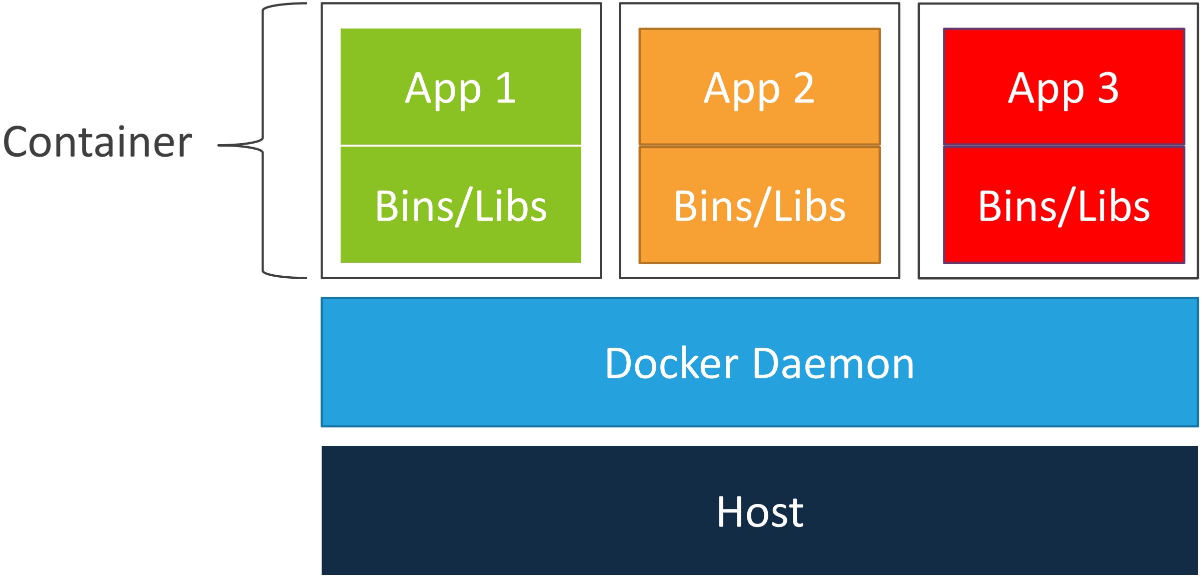


(a) Image dependency graph example

(b) Vulnerability propagation results

Inter-image dependency analysis example

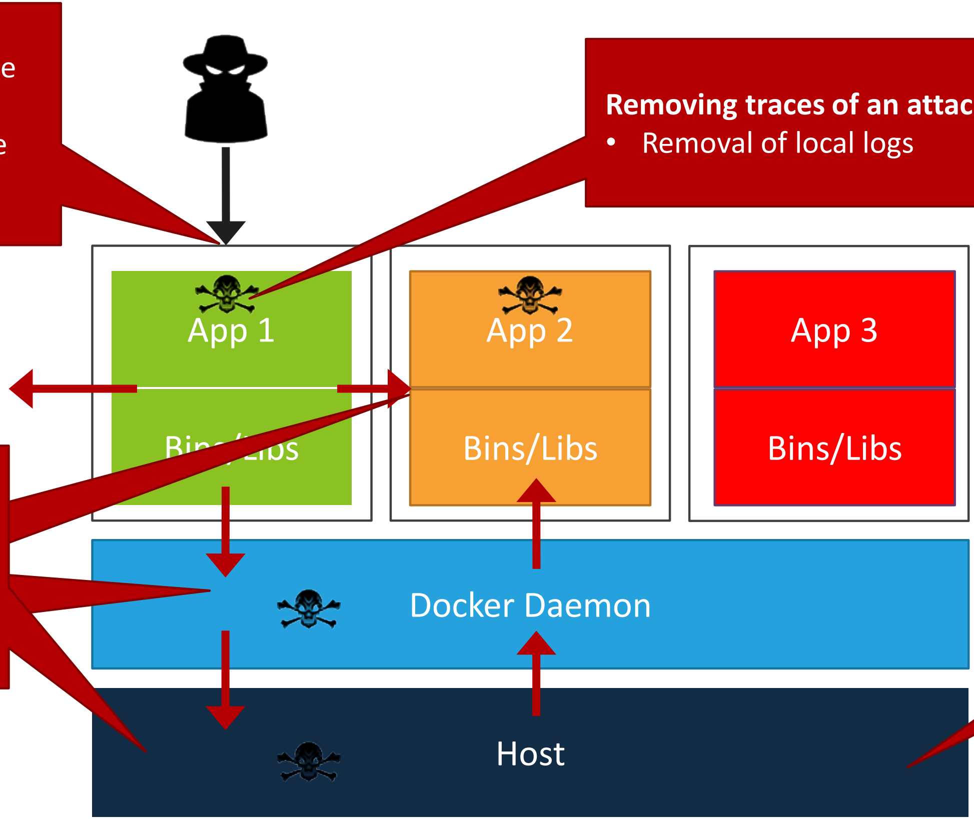
Docker architecture



Possible attack vectors

Getting into a container
- Software from unreliable sources
- Old versions of software
- A vulnerability in the application

Removing traces of an attack
- Removal of local logs



Extending an attack
- To other hosts (network)
- To other containers
- To Docker daemon
- To host

Negatively affecting other services
- Slowing down the server / eating resources
- Crashing the server

Docker security

How does Docker handle security?

- Kernel namespaces
- Control Groups
- Kernel capabilities
- Isolated file system (base image + [writable] container)
- Apparmor, seccomp
- Ulimits (in container startup, or global per-container config)
- User namespaces (map uids inside containers to an uid-namespace outside containers)

Kernel namespaces

- Linux kernel feature for isolating and virtualizing system resources
- When a container is started, Docker creates a set of namespaces for that container. Processes inside a container see only these namespaces (and no system artifacts)
- Examples: pid (process isolation), net (network isolation), ipc (interprocess communication), mnt (mount points), uts (unix timesharing system)
- Namespace support in Linux kernel since 2008, tested and mature code

Control groups (cgroups)

- Linux kernel feature. In kernel mainline since 2008
- Possibility to limit, account and isolate resource usage
- Applied when starting a container (docker run flags, or in docker-compose file)
- CPU, memory, max pids count, (network, disk I/O)

Kernel capabilities

- Traditional UNIX systems have privileged processes (uid 0, root) and unprivileged processes (uid != 0, non-root). Root processes bypass all kernel permission checks
- From kernel 2.2. onwards, root permissions can be split into more gradual list of capabilities
- In practice, if one gets into a container, limited capability possibilities make it harder to extend an attack
- Docker grants by default: SETPCAP, MKNOD, AUDIT_WRITE, CHOWN, NET_RAW, DAC_OVERRIDE, FOWNER, FSETID, KILL, SETGID, SETUID, NET_BIND_SERVICE, SYS_CHROOT, SETFCAP
- Not granted, for example: SYS_TIME, SYS_RAWIO, NET_ADMIN, SYS_PTRACE
- http://man7.org/linux/man-pages/man7/capabilities.7.html

A shared Kernel

- Host and Docker containers share the same kernel
- **Risk factor:** if the kernel contains a vulnerability, and code in a container can access it, easy to extend an attack
- Important to keep the kernel updated

Increasing Docker security

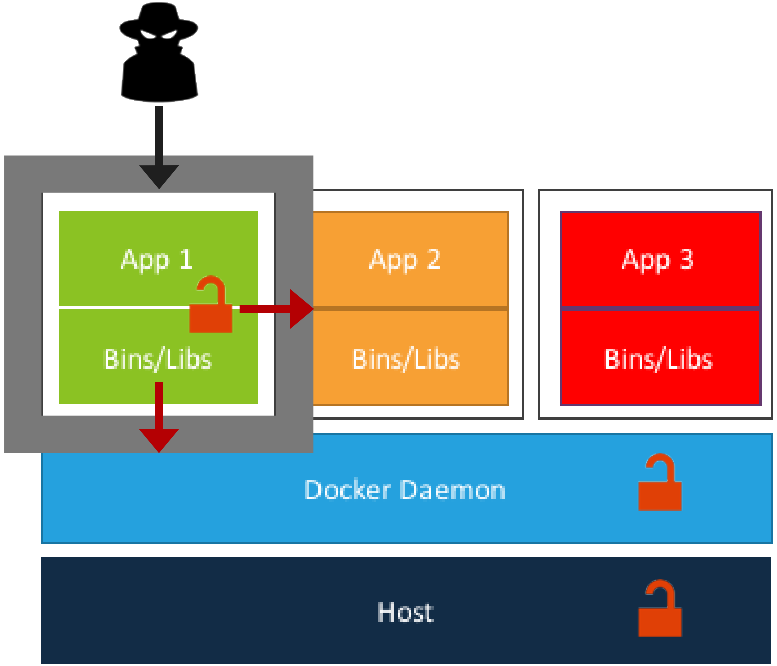
Different layers of security

- Docker image building (e.g. Dockerfile and processes)
- Docker runtime (docker run, docker-compose or similar)
- Docker Engine
- Docker host

Docker image hardening

What do we want to achieve?

- Limit the possibility of getting into a container
- Limit tools and possibility of using external tools for extending the attack
- Have a standardized way for creating and maintaining images



Docker image / tech recommendations

- Do not run software as root. Create an user instead (or use user namespaces)

- Prepare software so that root is mounted as read-only (and use tmpfs with limits for run files)

- Do not trust community images (even with public Dockerfile) on Docker hub. Build your images on official base images

- Build always on a fresh base image (e.g. docker pull [image] before build)

- Use minimal base image (for example alpine)

- When downloading software, check for checksums

Docker image / tech recommendations cont.

- Use specific versions (e.g. “FROM node:7.7.2-alpine instead of node:latest)

- Do not store secrets to Dockerfiles. Use docker secrets instead (ENV –variables are a bad practice, may leak information)

- Add a HEALTHCHECK command for orchestration

- Do not install unnecessary software (e.g. for debugging or testing purposes)

Docker image / policy recommendations

- Create hardened docker-compose.yml & Dockerfile templates to be distributed for software projects
- Review changes to Dockerfiles by a security/ops-knowledgeable person
- Make sure that when image is built later on, it’ll be exactly the same as before
- Use a CI pipeline to build Docker images
- Install a system to scan for vulnerabilities at Docker images (ecosystem still partially forming, multiple tools)

Docker runtime

- Use docker-compose instead of manual docker run commands

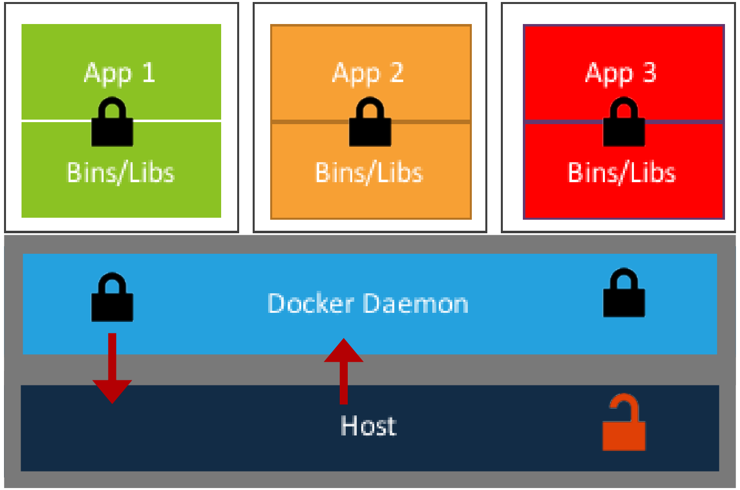
- Multiple benefits; e.g. container linking, private network generation

- Add default flags: drop unnecessary capabilities, limit new privileges (no-new-privileges), set memory limit, limit cpu usage when needed, set read-only flag

```
version: '2.1'
services:
  mongo-test:
    image: mongo:3.4.4
    security_opt: ["no-new-privileges"]
    cap_drop: ["all"]
    cap_add: ["SETUID", "SETGID", "CHOWN"]
    mem_limit: 256m
    cpu_shares: 1024
    read_only: true
    tmpfs:
    - "/run:rw,noexec,nosuid,size=128k"
    - "/tmp:rw,noexec,nosuid,size=10M"
    volumes: ["/srv/mongo-data-test:/data"]
```

Docker host & engine recommendations

- Keep host kernel updated!
- Use centralized logging with Docker log drivers (remote syslog, splunk, gelf, etc)
- Deny internal container communications (icc=false)
- Keep Docker updated
- Note that users who control docker daemon (belong to docker group) effectively have a root on host



digia

Apparmor & seccomp

- Linux Kernel security features, good for enhanced security. Supported by Docker since 2014 (apparmor) and 2016 (seccomp)
- Benefits: alleviate the risk of getting into a container, reduce the risk of extending an attack
- Still a bit of hassle to set up. Seccomp not available in Swarm mode (see moby#25209) or in Kubernetes (kubernetes feature #135). Kubernetes has beta-level apparmor support

Apparmor

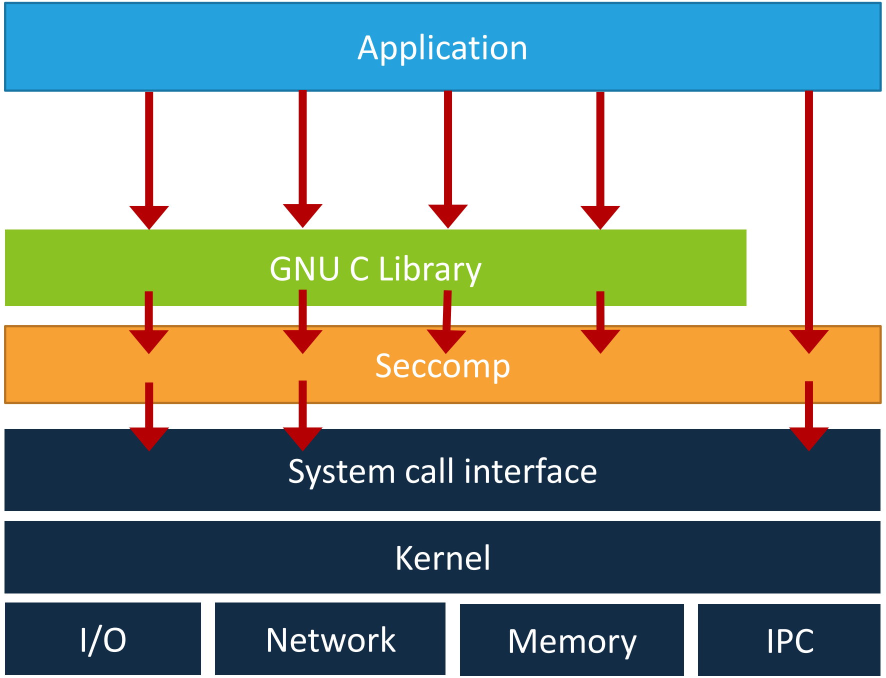
- App-specific profiles that restrict program capabilities such as file permissions and network access
- Initial release 1998 by SUSE, supported by Canonical since 2009. Not enabled by default in RedHat based distros

Seccomp

- seccomp = SECure COMPuting with filters
- Allows filtering of kernel syscalls that an application can make
- By Andrea Arcangeli, 2005. Available by default in most Linux systems

Seccomp

- Mitigates the risk of shared kernel between host and containers
- Limit the available syscalls only to the ones needed by a container
- If a process in a container accesses denied syscall, it'll get SIGKILL
- Profile is in JSON format. Use strace to get list of all syscalls



digia

Seccomp (cont.)

- Docker has a default seccomp profile that limits some available syscalls
- $ docker run –security-opt no-new-privileges –security-opt seccomp=profile.json hello-world
- Preferably in docker >=1.13, might need to add docker-specific syscalls in lower versions
- See moby/moby repo issues #22252, #24661

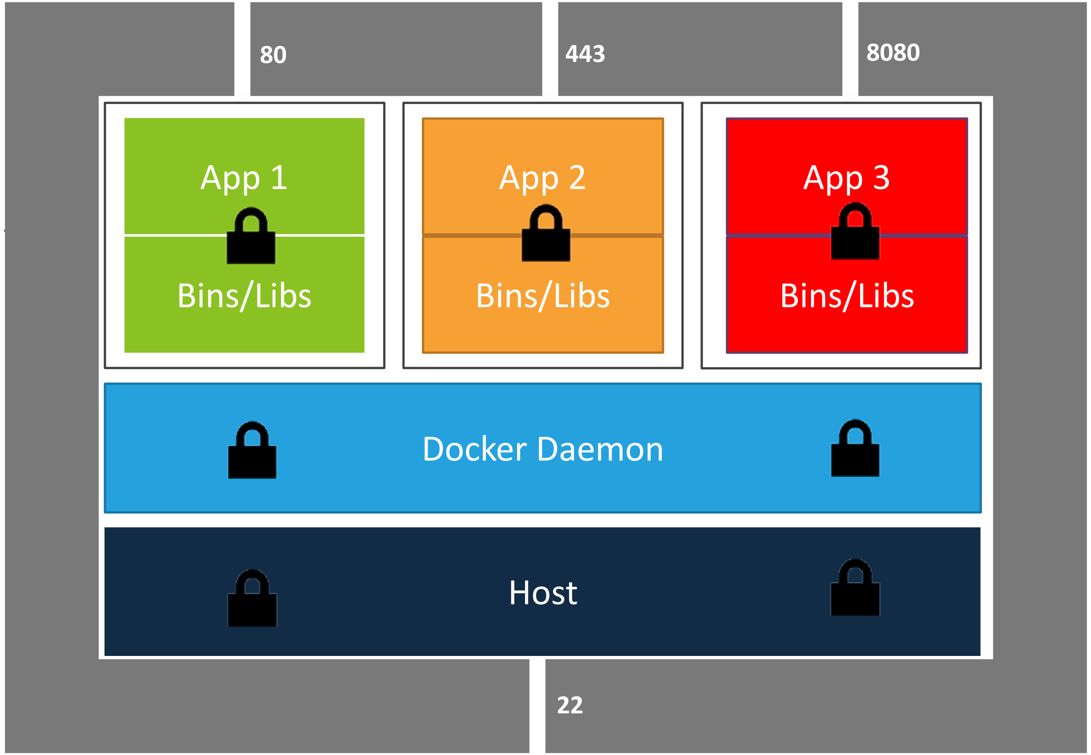
Apparmor

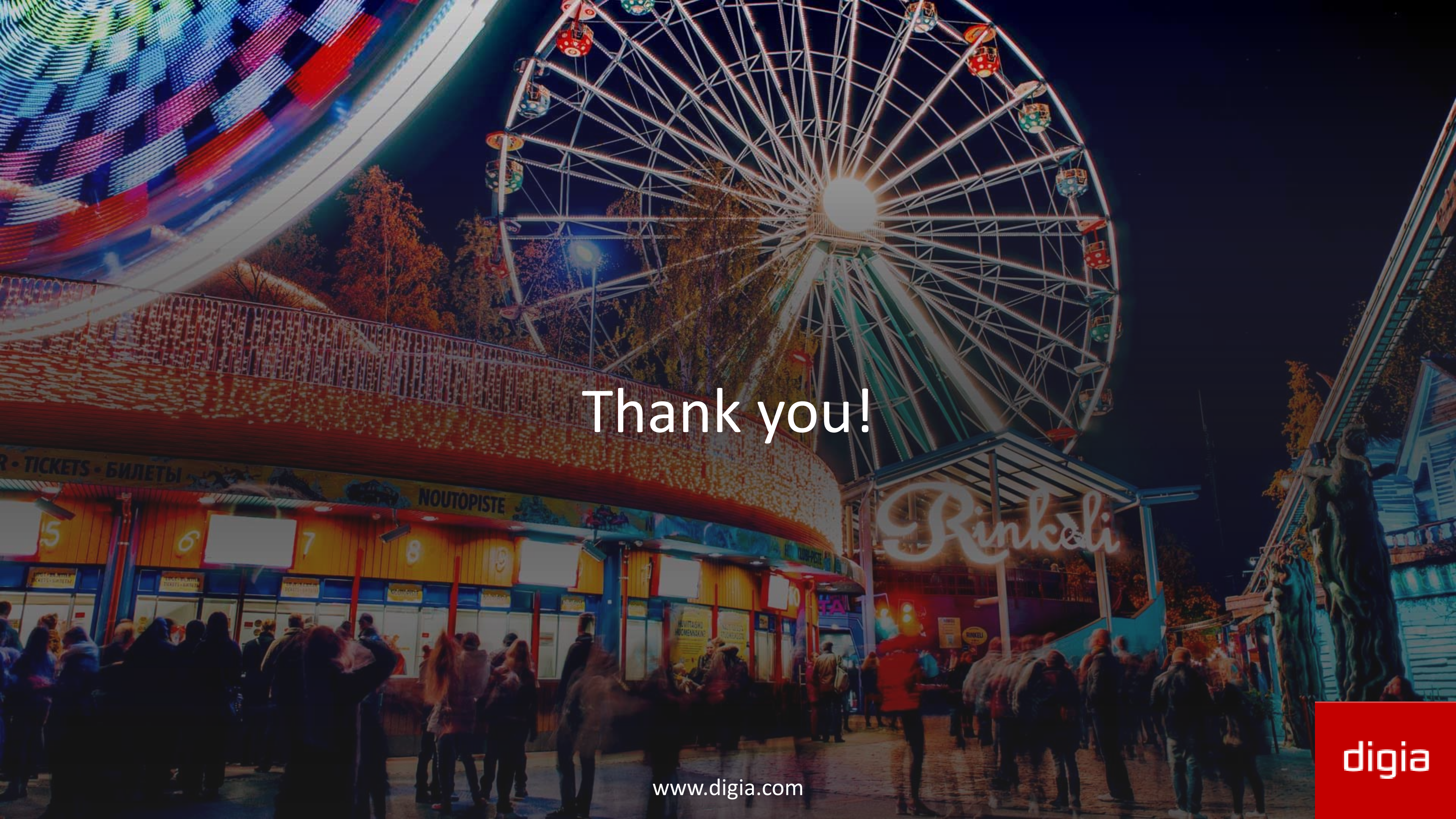
- Mostly in Debian based OS’es
- Used mainly for per-file permission limits
- r = read, w = write, a = append, x = execute, m = memory map executable, k = lock, l = link
- Prepend a line with “owner” keyword to only allow UID of the process

```
profile docker-nginx
flags=(attach_disconnected,mediate_deleted) {
  /etc/ld.so.cache r,
  /etc/nginx/conf.d r,

  /run/nginx.pid rw,
  /var/cache/nginx/** rw,

  … etc
}
```


80
443
8080
App 1
Bins/Libs
App 2
Bins/Libs
App 3
Bins/Libs
Docker Daemon
Host
22



Thank you!

www.digia.com