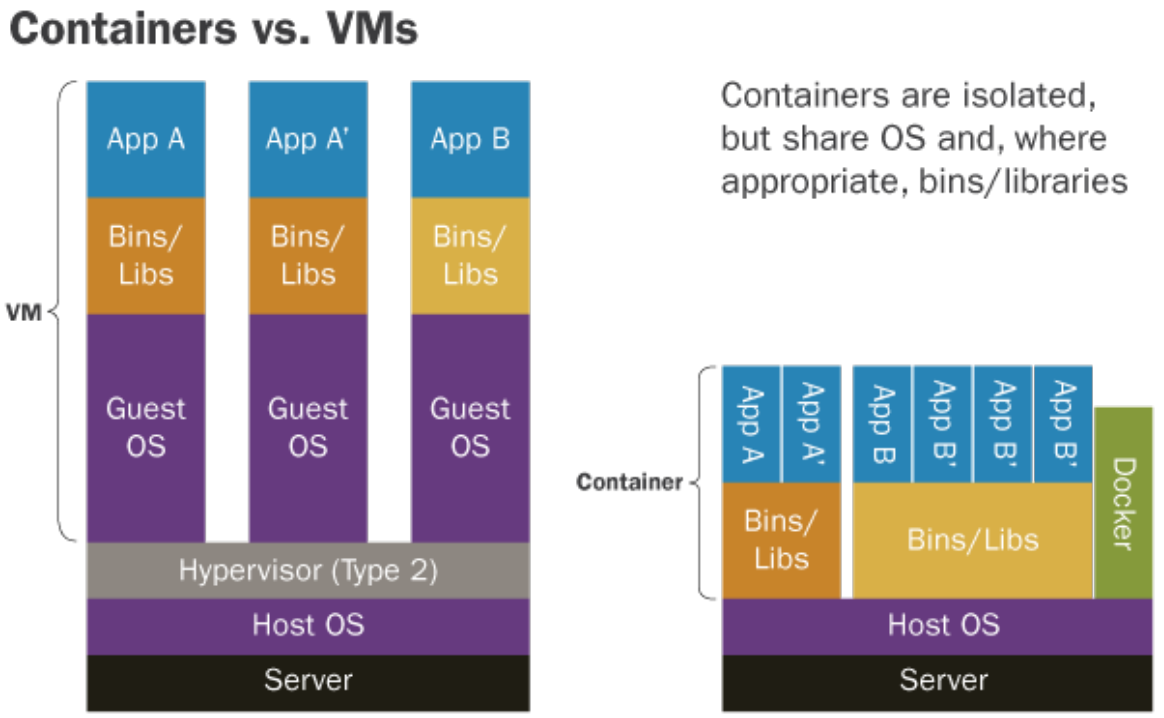
Securing Docker Deployments

Advancements, Considerations and Best Practices

Paris Zoumpouloglou
zubu.re
@pzubu

Docker in 60 seconds

- Initial release March 2013
- Developed in Go
- Complete software ecosystem around Linux containers.
- Less overhead, isolation (?), better resource management



Docker in 60 seconds

- Linux kernel resource isolation capabilities
  - **Cgroups** – Resource Management (CPU, Memory, net)
  - **Namespaces** – Process Isolation
  - **Shared kernel** for host and containers
- Software distribution using Docker Images (Docker Hub)
- Many similarities with Git workflow

```
docker pull <image name>
docker commit <container>
...
```

Docker in 60 seconds

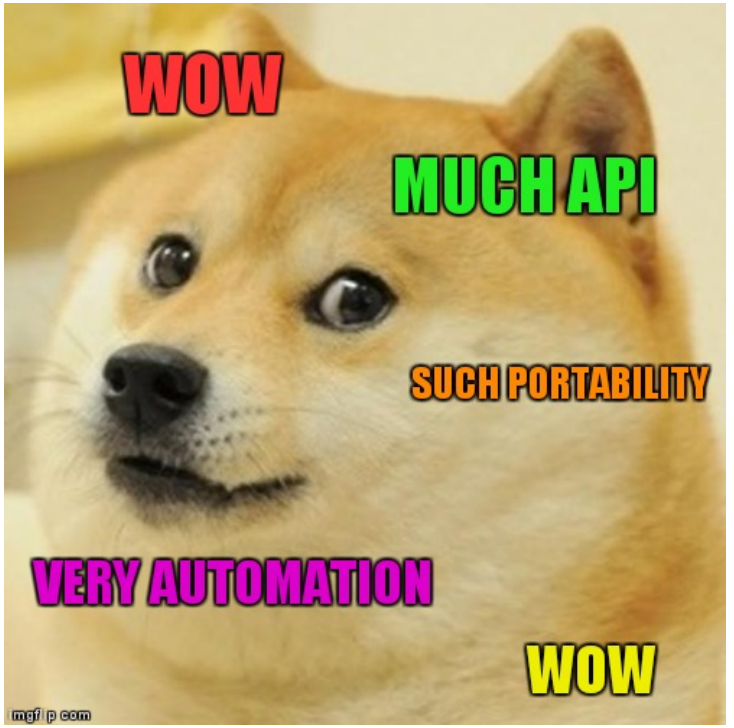
- Backed by major industry players (Amazon, Microsoft, IBM etc.)
- Era of CI/CD
- The eternal Linux “works on my machine” struggle
- Hype! Hype! Hype!

Docker Attack Surfaces

- Docker Daemon
  - One service to rule them all
- Containers
  - Containers do not (always) contain!
- Image Distribution
  - Public Docker Images anyone?

First things first

```
$ curl -fsSL https://get.docker.com/ | sh
```



Docker may (but shouldn't). You 're not Docker. **Just don't**.

Who controls the daemon controls the host

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

- Why not add users to 'root' group then?!
- 2-liner privilege escalation (source)

```
docker run -v $PWD:/stuff -t my-docker-image /bin/sh -c \

'cp /bin/sh /stuff && chown root:root /stuff/sh && chmod
a+s /stuff/sh'
```

- [CVE-2014-3499] systemd socket activation results in privilege escalation (packaging bug, world `rw` socket)
- **Docker 1.10** introduced Authorization Plugins
  - Granular access policies!

Syscalls! Syscalls! Syscalls!

- Linux kernel has 300+ syscalls
  - syscall → potential attack surface
- Enter `seccomp`!
  - Linux kernel security feature. Introduced in **Docker 1.10**
  - Allows a process to specify a Berkeley packet filter to syscalls
  - Default profiles available!

Kernel Capabilities

- Linux divides the privileges traditionally associated with superuser into distinct units, known as **capabilities**
- e.g. bind to < 1024 port is `net_bind_service` cap
- Docker drops most “dangerous” capabilities, e.g.:
  - `CAP_SYS_RAWIO` - Modify kernel memory
  - `CAP_SYS_MODULE` - Insert and remove kernel modules
  - Etc.
- Containers can run with `—cap-add` or `—cap-drop` options.

  Use wisely!

Containers do not (always) contain

- [CVE-2015-3627] Insecure opening of file-descriptor 1 leading to **privilege escalation**
- [CVE-2015-3629] Symlink traversal on container respawn allows local **privilege escalation**
- [CVE-2015-3630] Read/write proc paths allow **host modification** & information

Namespaces

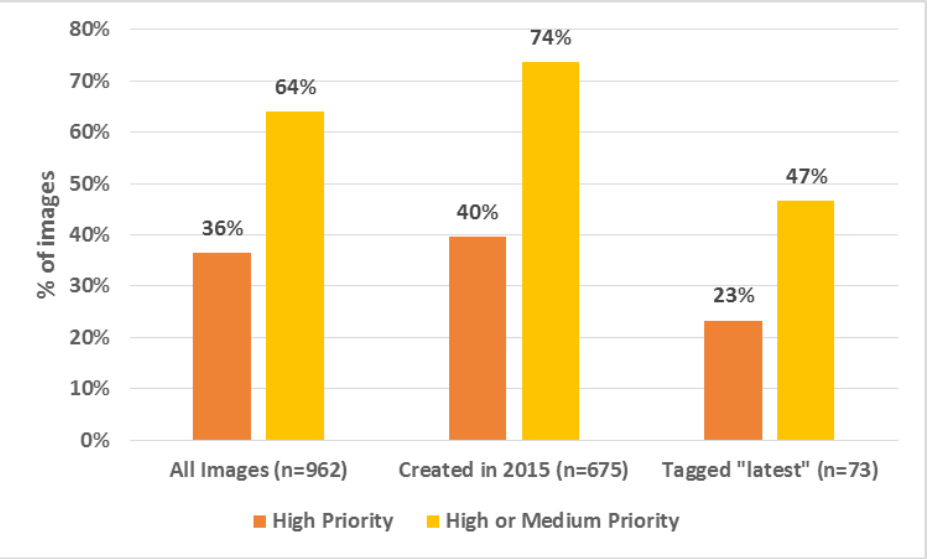
Namespaces are a Linux kernel feature that isolates and virtualizes resources (PID, hostname, userid, network, ipc, filesystem) of a collection of processes.

- LXC abstraction until 0.9, switched to libcontainer (Go)
  - Fewer moving parts, consistency
- libcontainer did not support user namespaces until recently
  - container root == host root (hint: breakout)
  - Before v1.0 → container root was the only option
- **Docker 1.10** introduced user namespaces!
  - container root != host root

What about Docker images?

Someone said that 30% of the images on the Docker Registry contain vulnerabilities (source)

- Trust but verify
- Look out for outdated images in the hub
- Lots of advancements starting from 1.8 (Docker Content Trust)
- Nautilus Project




For all you know they were
made by Russian hackers!

So...?

- `apt-get remove —purge-with-fire docker`?
  - NO!
- Containers are here to stay. Why?
  - Great for packaging
  - Ultra-fast deployments
  - Unikernels might be a thing soon.
- Security people don't take ~~change~~ hype well, hipsters do.

Docker Hardening

- Lots of options, many insecure by default
- Be smart, use docker-bench-security

```
➜ docker-security-benchmark git:(master) docker run -it --net host --pid host -v /var/run/docker.sock:/var/run/docker.sock \
> -v /usr/lib/systemd:/usr/lib/systemd -v /etc:/etc --label security-benchmark \
> diogomonica/docker-security-benchmark
# ------------------------------------------------------------------------
# CIS Docker 1.6 Benchmark v1.0.0 checker
#
# Docker, Inc. (c) 2015
#
# Provides automated tests for the CIS Docker 1.6 Benchmark:
# https://benchmarks.cisecurity.org/tools2/docker/CIS_Docker_1.6_Benchmark_v1.0.0.pdf
# ------------------------------------------------------------------------
Initializing Thu May 14 10:37:29 PDT 2015


[INFO] 1 - Host Configuration
[WARN] 1.1  - Create a separate partition for containers
[PASS] 1.2  - Use an updated Linux Kernel
[WARN] 1.5  - Remove all non-essential services from the host - Network
[WARN]      * Host listening on: 6 ports
[PASS] 1.6  - Keep Docker up to date
[INFO] 1.7  - Only allow trusted users to control Docker daemon
[INFO]      * docker:x:999:
```

… and keep an eye for Actuary (WIP)

Few words on Actuary

- Docker-bench-security successor
- Written in Go (previously bash)
- 50+ security checks
- Supports custom audit profiles (previously not supported)
- Logging capabilities
- Web service providing official profiles (e.g for AWS) and profile generation.

SELinux / AppArmor

- Both supported by Docker containers
- Process/resource isolation policies
- SELinux provides more control than AA
- Want to use AppArmor?
  - Try bane by @jfrazelle!
- Want to use SELinux?
  - It is worth the time but no easy way around.

    Good luck!

Bane

- Human-readable TOML profiles

```
$ sudo bane sample.toml

# Profile installed successfully you can now run the profile with

# `docker run --security-opt="apparmor:docker-nginx-sample"`
```

```
# name of the profile, we will auto prefix with `docker-`
# so the final profile name will be `docker-nginx`
Name = "nginx-sample"

[Filesystem]
# read only paths for the container
ReadOnlyPaths = [
        "/bin/**",
        "/boot/**",
        "/dev/**",
        "/etc/**",
        "/home/**",
        "/lib/**",
        "/lib64/**",
        "/media/**",
        "/mnt/**",
        "/opt/**",
        "/proc/**",
        "/root/**",
        "/sbin/**",
        "/srv/**",
        "/tmp/**",
        "/sys/**",
        "/usr/**",
]

# paths where you want to log on write
LogOnWritePaths = [
        "/**"
]

# paths where you can write
WritablePaths = [
        "/var/run/nginx.pid"
]

# allowed executable files for the container
AllowExec = [
        "/usr/sbin/nginx"
]

# denied executable files
DenyExec = [
        "/bin/dash",
        "/bin/sh",
        "/usr/bin/top"
]
```

Container/Image Visibility

- Every container may run a different version of the same software
- Lots of OSS tools for analysis
  - Banyan
  - Clair
  - OpenSCAP

Containers/Images in production

- Use private Docker registries
- Use only official images
- Use TLS/SSL
- Remove unused/old images
- Install only necessary packages
- Enable Content Trust

Final Words

- “- DevOps, meet InfoSec.”
- Container isolation is much thinner than traditional Vms. Treat it that way.
- Own your prod. images, Docker won't do that for you!
- Docker security is becoming seriously more mature (lots of accessible tools too!). Contribute?

Thank you!
Questions?