Docker London

July 20, 2016

Container Security

Everything You Probably Should Know

...but most of which I'm neither an expert on nor could we ever cover in the time allotted...

Who am I?

(skipping the metaphysical aspects)

Phil Estes
Senior Technical Staff Member
IBM Cloud, Open Technologies
Container Strategy/Open Source Leader

Docker community core engine maintainer <
Linux/open source tech. @ IBM for 12 yrs <

Community activities & accomplishments
> User namespace support in the Docker engine
> Design of v2.2 image specification
> Implemented multi-platform image tool
> Member of the "Docker Captains" program

"...This was largely by an effort of IBM's Phil Estes (although he debates that effort)*"

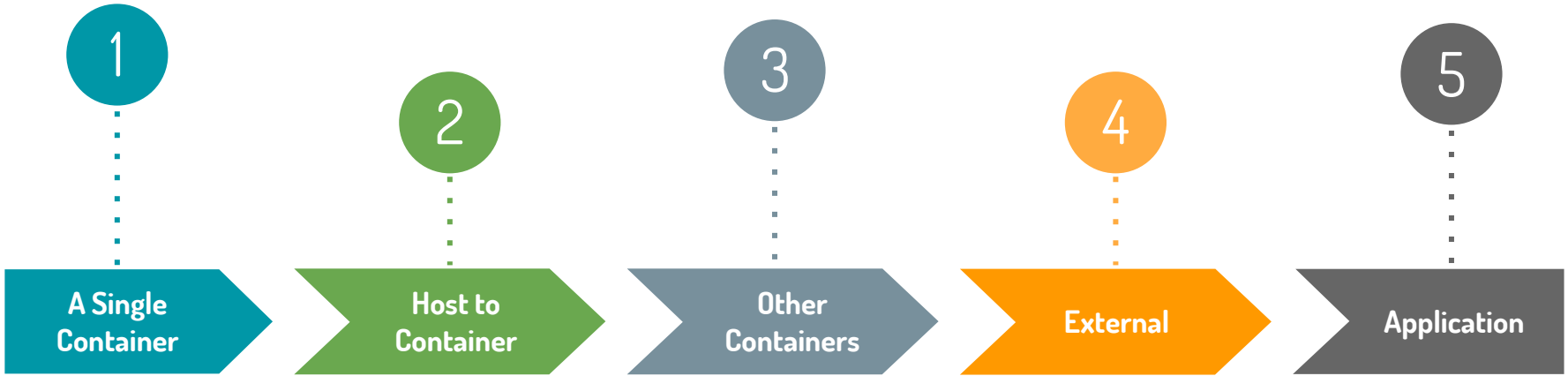
**NCC Group report "Understanding and Hardening Linux Containers", p. 68, section 8.1.4*

What are we going to cover?

I mean, security is a **huge** topic!

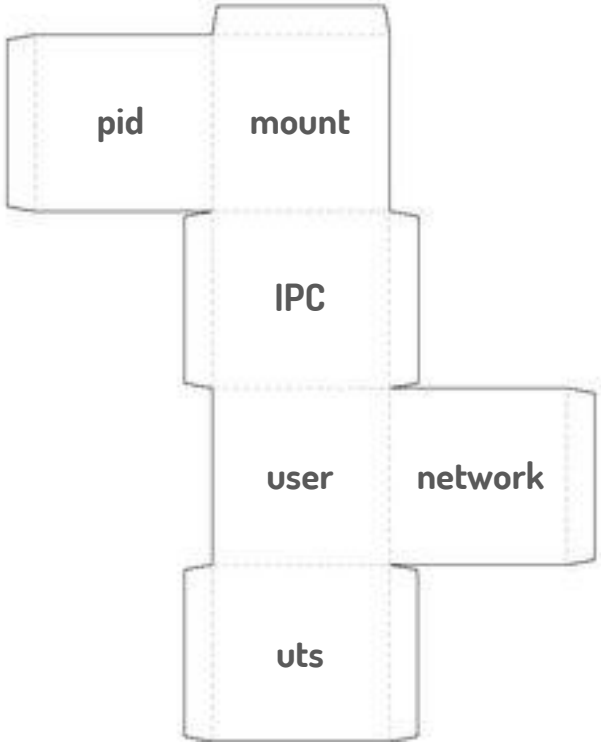
THREATS

Security is mostly a war against **threats**, so we might as well try and look at the **threat "vectors"** that affect the world of containers. We can call the areas of weakness our "**attack surface**" with our main goal being to reduce the attack surface in each of these areas. Most importantly we need to agree that these threats are not hypothetical and **will happen** at some point. Hence our need to consider security as **important** as any other topic we discuss around our application lifecycle.



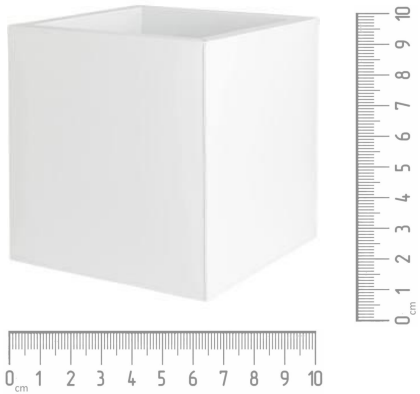
Basics: What is a Container?

What is it we're trying to secure?



> Linux kernel **namespaces** provide the **isolation** (hence "container") in which we place one or more processes

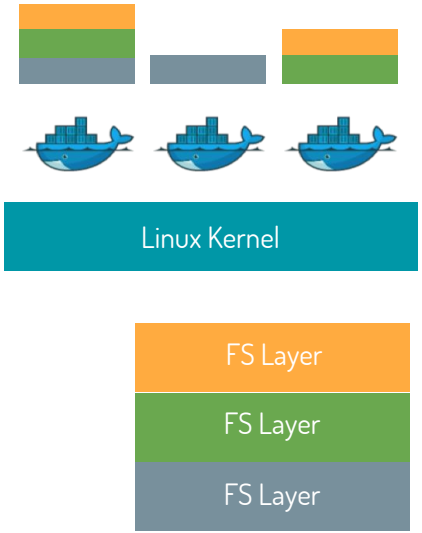
> Linux kernel **cgroups** ("Control groups") provide **resource limiting** and accounting (CPU, memory, I/O bandwidth, etc.)



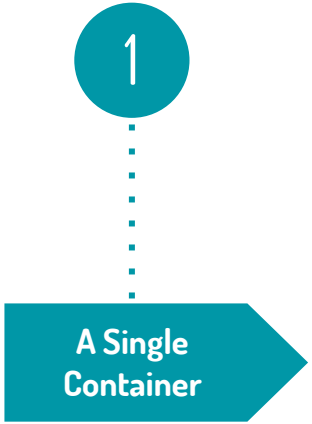
Container Properties

Our definition, continued

- A **shared kernel** across all containers on a single host
- **Unique filesystem** that could look like a Linux distro, but need not be
  - With Docker, this is a layered model where, using CoW (copy-on-write) union filesystems we all can share a set of underlying read-only content (writes happen on an unshared “top” layer)
- Linux namespaces are **shareable** (see Kubernetes “pod” concept); so containers do not have to have explicit 1-to-1 boundaries
- Ignoring for the moment Canonical/LXD “system containers” definition, application container models expect **one process per container**



Single Container



- Let’s consider the base security assumptions:
  - Reliance on **Linux kernel features** to properly **isolate** and **control resources** (trust that weaknesses and/or breakout scenarios are approaching zero)
  - Assume that contained processes are **well-behaved** and that code (binaries and libraries) accessible within contained environment is secure
  - Assume the code we are running is what we asked to run (signed/trusted image registry; tamper-proof image validation)

Host <-> Container

Protecting the host from containers

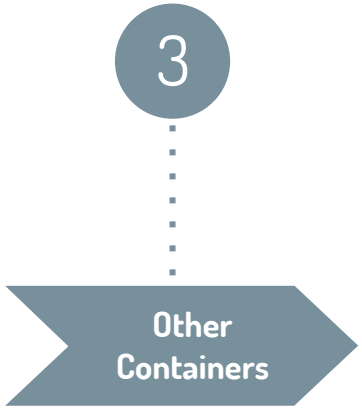
2

Host to Container

THREAT	MITIGATION
DoS Host (use up CPU, memory, disk), **Forkbomb**	Cgroup controls, disk quotas (1.12), kernel pids limit (1.11 + Kernel 4.3)
Access host/private information	Namespace configuration; AppArmor/SELinux profiles, seccomp (1.10)
Kernel modification/insert module	Capabilities (already dropped); seccomp, LSMs; don't run `--privileged` mode
Docker administrative access (API socket access)	Don't share the Docker UNIX socket without Authz plugin limitations; use TLS certificates for TCP endpoint configurations

Container <-> Container

Malicious or Multi-tenant



THREAT	MITIGATION
DoS other containers (noisy neighbor using significant % of CPU, memory, disk)	Cgroup controls, disk quotas (1.12), kernel pids limit (1.11 + Kernel 4.3)
Access other container's information (pids, files, etc.)	Namespace configuration; AppArmor/SELinux profile for containers
Docker API access (full control over other containers)	Don't share the Docker UNIX socket without Authz plugin limitations (1.10); use TLS certificates for TCP endpoint configurations

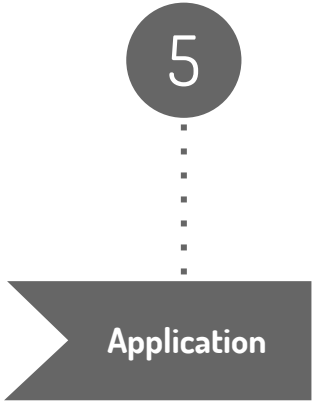
External -> Container

The big, bad Internet

THREAT	MITIGATION
DDoS attacks	Cgroup controls, disk quotas (1.12), kernel pids limit (1.11 + Kernel 4.3) Proactive monitoring infrastructure/operational readiness
Malicious (remote) access	Appropriate application security model No weak/default passwords! --readonly filesystem (limit blast radius)
Unpatched exploits (underlying OS layers)	Vulnerability scanning (IBM Bluemix, Docker Data Center, CoreOS Clair, Red Hat “SmartState” CloudForms (w/Black Duck)

Application Security

New problem; same as the old problem



THREAT

No specific attack surface unique to containers (same application security issues as VMs, bare metal clouds)

MITIGATION

Significant container benefit: provided protections are in place (seccomp, LSMs, dropped caps, user namespaces) the exploited application has greatly reduced ability to inflict harm beyond container "walls"

- Proper handling of secrets through dev/build/deploy process (no passwords in `Dockerfile`, as an example)
- Unnecessary services not exposed externally (shared namespaces; internal/management networks)
- Secure coding/design principles

Your Docker Security Toolbox

A closer look at what's available

Cgroups

Control/limit container access to CPU, memory, swap, block IO (rates), network

LSMs

AppArmor and SELinux are both supported in the Docker engine (via runc); a default profile is applied for the engine and containers

Capabilities

Docker by default only allows 14 of the 37 Linux capability groups; more can be dropped or added as required

Seccomp

Fine grained per-syscall control is available via seccomp; a default profile limiting many syscalls is already applied

Userns

User namespaced processes remap root to an unprivileged ID on the host. Docker supports a global uid/gid mapping

BUT WAIT, THERE'S MORE!

--pids-limit for controlling PID limitations per container (forkbomb prevention); **--no-new-privileges** to prevent privilege escalation, **--readonly** filesystem for immutable container image; **DOCKER_CONTENT_TRUST=1** for notary/signed image provenance, **Authz** plugins (Twistlock), **TLS** certificate-based API endpoint configuration; **Storage quotas** for specific Docker storage backends (btrfs, zfs in 1.12; devicemapper already available)

Cgroups

Limit resource use

Example: Use cgroups to set a memory limit

Other options: --kernel-memory, --memory, --memory-swap, --cpu-period, --cpu-quota, --cpu-shares, --cpuset-cpus, --cpuset-mems, --device-read-bps, --device-read-iops, --device-write-bps, --device-write-iops, --blkio-weight, --blkio-weight-device

```
$ docker run -m 32m estesp/hogit:latest
<output ends after a few iterations; pid exit code 137>

$ docker stats
<note memory use climbing up to 32MB>

$ docker inspect -f ' {{.State.OOMKilled}} ' <containerID>
 true

$ docker inspect -f ' {{.HostConfig.Memory}} ' <containerID>
 33554432
```

LSMs

AppArmor/SELinux

Example: Limit access to specific filesystem paths in container
Resources: https://github.com/jfrazelle/bane

```
$ sudo bane sample.toml
<creates new apparmor profile and installs it>

$ docker run --rm -ti --security-opt="apparmor:docker-nginx-sample" \
         -p 80:80 nginx bash
root@6da5a2a930b9:/# top
bash: /usr/bin/top: Permission denied
root@6da5a2a930b9:/# touch ~/thing
touch: cannot touch 'thing': Permission denied
```

Capabilities

Add/Drop Linux Kernel Capabilities

Example: Drop unnecessary capabilities from a container
Resources: http://man7.org/linux/man-pages/man7/capabilities.7.html

```
$ docker run --rm -ti busybox sh
/ # hostname foo
hostname: sethostname: Operation not permitted

$ docker run --rm -ti --cap-add=SYS_ADMIN busybox sh
/ # hostname foo
<hostname changed>

$ docker run --rm -ti --cap-drop=NET_RAW busybox sh
/ # ping 8.8.8.8
ping: permission denied (are you root?)
/ #
```

Seccomp

Linux Secure Computing

Example: Block specific syscalls from being used by container binaries

Resources: https://github.com/docker/docker/blob/master/docs/security/seccomp.md
http://blog.aquasec.com/new-docker-security-features-and-what-they-mean-seccomp-profiles

```
$ cat policy.json
{
    "defaultAction": "SCMP_ACT_ALLOW",
    "syscalls": [
        {
            "name": "chmod",
            "action": "SCMP_ACT_ERRNO"
        }
    ]
}
$ docker run --rm -it --security-opt seccomp:policy.json busybox chmod 640
/etc/resolv.conf
chmod: /etc/resolv.conf: Operation not permitted
```

User Namespaces

Linux user namespace support

Example: Enable user namespaces on the Docker daemon for all containers
Resources: http://man7.org/linux/man-pages/man7/user_namespaces.7.html
https://integratedcode.us/2016/02/05/docker-1-10-security-userns/

```
$ docker daemon --userns-remap=default(|someuser:somegrp)
<daemon starts with uid and gid mappings from /etc/sub{u,g}id>

$ docker run --rm -ti -v /bin:/host/bin busybox sh
/ # cp mybadshell /host/bin/sh
cp: can't create '/host/bin/sh': File exists

/ # cd /host/bin && mv sh sh.bak
mv: can't rename 'sh': Permission denied

/ #
```

Docker: Secure Out of the Box

Aiming for secure-by-default with ease of use

- Users/packagers won't turn on security if it's difficult (AppArmor profiles are hard to write; SELinux can be even harder)
- **Sane defaults** are tricky as well - someone's app won't work and they **will** complain
- Docker painstakingly tries to find a balance (e.g. DCT off by default, allowance for insecure registries)

** NCC Group report "Understanding and Hardening Linux Containers", v1.1, p. 97, section 9.13*

Available Container Security Features, Requirements and Defaults			
Security Feature	**LXC 2.0**	**Docker 1.11**	**CoreOS Rkt 1.3**
User Namespaces	Default	Optional	Experimental
Root Capability Dropping	Weak Defaults	Strong Defaults	Weak Defaults
Procfs and Sysfs Limits	Default	Default	Weak Defaults
Cgroup Defaults	Default	Default	Weak Defaults
Seccomp Filtering	Weak Defaults	Strong Defaults	Optional
Custom Seccomp Filters	Optional	Optional	Optional
Bridge Networking	Default	Default	Default
Hypervisor Isolation	Coming Soon	Coming Soon	Optional
MAC: AppArmor	Strong Defaults	Strong Defaults	Not Possible
MAC: SELinux	Optional	Optional	Optional
No New Privileges	Not Possible	Optional	Not Possible
Container Image Signing	Default	Strong Defaults	Default
Root Interation Optional	True	False	Mostly False

Container Security Futures

Looking into the crystal ball

- Fully unprivileged containers
  - Non-root user can execute container runtime without escalated/root privilege
  - Significant activity and experiments in recent months; some challenges to overcome
- Image signing/provenance (Docker Content Trust) on by default
- User namespaces phase 2: custom namespaces ranges per container
  - Upstream kernel support for uid/gid file ownership shift
  - Allows for multi-tenant cloud to provide uid/gid maps per tenant with no overlap
- Network security
  - Docker 1.12 - overlay with IPSec over vxlan with “-o secure”; control plane already encrypted

Resources

Where to find more information

> NCC Group Report "**Understanding and Hardening Linux Containers**" v1.1

Author: **Aaron Grattafiori** (@dyn___ on Twitter)
https://www.nccgroup.trust/us/our-research/understanding-and-hardening-linux-containers/

> Docker Security **Online Documentation**

Author: **Docker contributors/maintainers**
https://docs.docker.com/engine/security

> **CIS Docker 1.11.0 Benchmark** v1.0.0 (April 2016)

Author: **Center for Internet Security**
https://benchmarks.cisecurity.org/tools2/docker/CIS_Docker_1.11.0_Benchmark_v1.0.0.pdf

THANK YOU!

@estesp

github.com/estesp

estesp@gmail.com

https://integratedcode.us

IRC: estesp

Credits

Shapes & Icons

Vectorial Shapes in this Template were created by **Free Google Slides Templates** and downloaded from **pexels.com** and **unsplash.com**.

Icons in this Template are part of Google® Material Icons and **1001freedownloads.com**.

Fonts

The fonts used in this template are taken from **Google** fonts. (Dosis,Open Sans)
You can download the fonts from the following url: https://www.google.com/fonts/

Backgrounds

The backgrounds were created by **Free Google Slides Templates**.

Color Palette

The Template provides a theme with four basic colors:

#93c47dff	#0097a7ff
#78909cff	#eeeeeeff
#f7b600ff	#00ce00e3
#de445eff	#000000ff

Trademarks

Microsoft® and PowerPoint® are trademarks or registered trademarks of Microsoft Corporation.

Google Drive® is a registered trademark of Google Inc.