

RSA®Conference2015

San Francisco | April 20-24 | Moscone Center

SESSION ID: CSV-T07R

Something Awesome on Cloud and Containers

Christofer Hoff

VP, Security CTO
Juniper Networks
@Beaker

Rich Mogull

Analyst and Chief Executive Officer
Securosis, LLC
@rmogull



RSAC®Conference2015

San Francisco | April 20-24 | Moscone Center

The following is an excerpt
from the presentation...
skipping to the bits on
Containers



RSACConference2015

San Francisco | April 20-24 | Moscone Center

Containers & Why They Matter

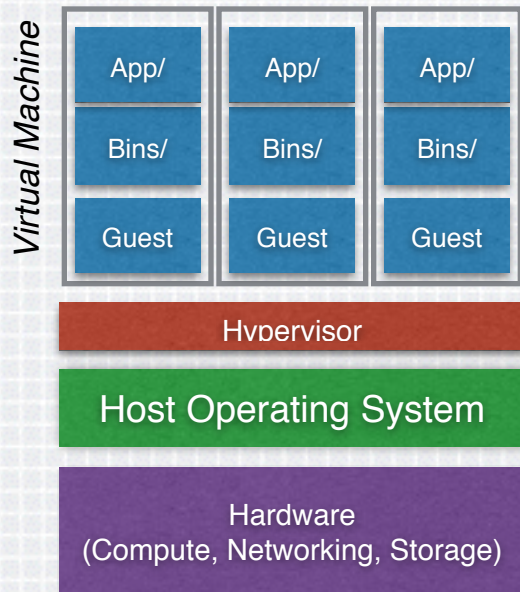


What are Containers? And LXC?

- ◆ “Containers” have existed in many forms for years: FreeBSD Jails, OpenVZ, Solaris Zones, and LXC for example.
- ◆ LXC (Linux Containers) is a userspace interface for the Linux kernel containment features to enable multiple isolated Linux processes to co-exist on a single Linux host
 - ◆ *“LXC containers are often considered as something in the middle between a chroot and a full fledged virtual machine. The goal of LXC is to create an environment as close as possible to a standard Linux installation but without the need for a separate kernel.”*
- ◆ This is enabled by the use of two Linux kernel process resource management solutions:
 - ◆ **Cgroups** (control groups) are a resource management solution providing a generic process-grouping framework which limits and prioritizes system resources (CPU, memory, I/O, network, etc.)
 - ◆ **Namespaces** allow for lightweight process virtualization and enables processes to have different views of the system (mnt, pid, net, pic, uts, user)



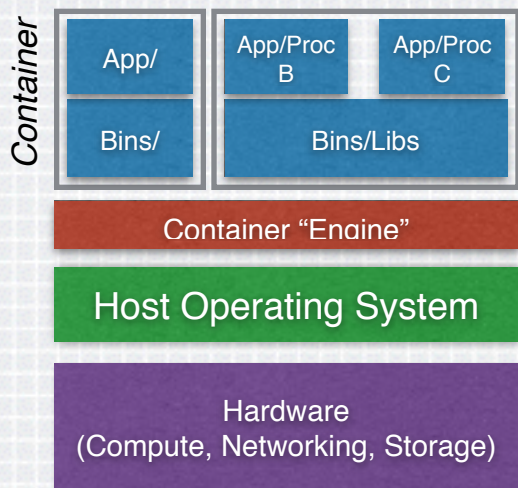
Comparing Virtual Machines...



- ◆ Virtual Machines (VMs) are best used to emulate and allocate chunks of hardware resources.
- ◆ The isolation enabled by hypervisors generally represent less of an attack surface than exposing the entire host OS to the VMs themselves and provides an abstracted/protected layer
- ◆ In the case of a Type-1 hypervisor, there is no underlying host OS
- ◆ Each VM includes not only the application - which may be only 10s of MB - and the necessary binaries and libraries, but also an entire guest operating system - which may entail many gigabytes of storage and must be patched independently



...and Containers?



- ◆ Containers operate at the process level, which makes them very lightweight and perfect as a unit of software delivery.
- ◆ A container comprises just the application and its dependencies. It runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers.
- ◆ Thus, it enjoys *many* of the resource isolation and allocation benefits of VMs but is much more portable and efficient.



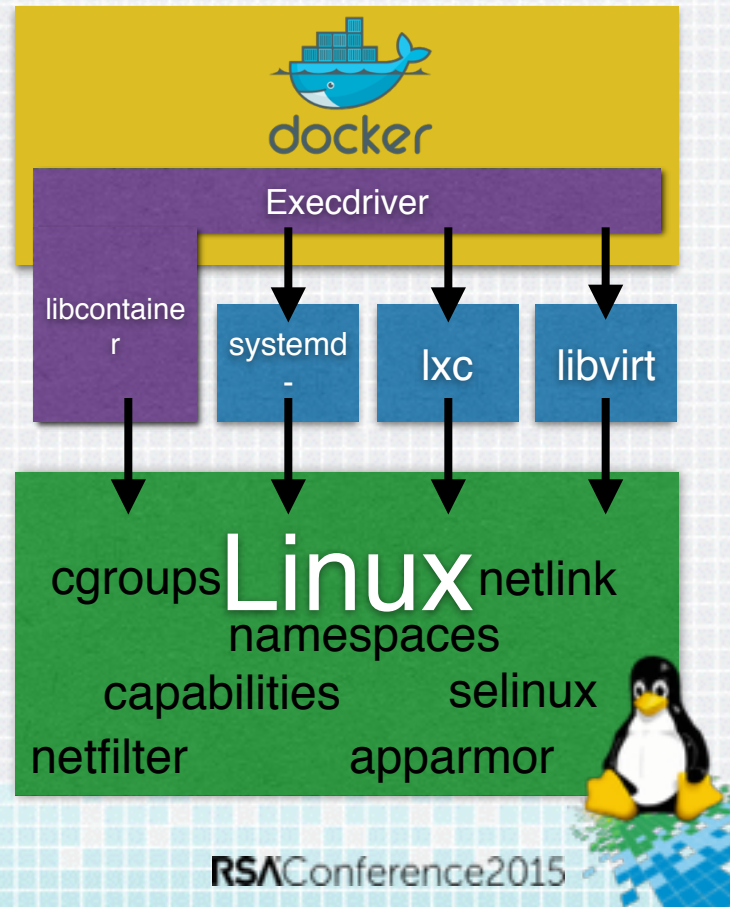
So What Is Docker? [non-technical]

- ◆ Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications.
- ◆ Consisting of Docker Engine, a portable, lightweight runtime and packaging tool, and Docker Hub, a cloud service for sharing applications and automating workflows, Docker enables apps to be quickly assembled from components and eliminates the friction between development, QA, and production environments.
- ◆ Docker is evolving into a platform that includes tools such as Machine, Swarm and Compose to enable simpler, integrated docker engine deployment, clustering, and distributed multi-application orchestration



What Is Docker [more technical]

- ◆ Docker is an open-source project written in Go, that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux.
- ◆ Docker implements a high-level API to provide lightweight containers that run processes in isolation, building on top of facilities provided by the Linux kernel (primarily cgroups and namespaces)
- ◆ A Docker container does not require or include a separate operating system. It relies on the kernel's functionality and uses resource isolation (CPU, memory, block I/O, network, etc.) and separate namespaces to isolate the application's view of the operating system.
- ◆ Docker accesses the Linux kernel's virtualization features either directly through the provided libcontainer library or indirectly via libvirt, LXC or systemd-nspawn.
- ◆ Libcontainer enables containers to manipulate Linux namespaces, control groups, capabilities, AppArmor security profiles, network interfaces and firewalling rules in a consistent and predictable way.



Are Containers “Secure?”



That is the wrong question.



So? TL;DR...



- ◆ As you will see, to adequately secure environments that utilize containers, you will be required to know more about the internals and hardening of the underlying host OS to ensure the integrity and security of said platforms **more** than you might with a hypervisor and VMs.
- ◆ **Or** you should/can find a PaaS/Container management solution that includes security capabilities so you don't have to.
- ◆ **Or** you should/can just “give up” and run your containers within VMs...



4 Dimensions of Container Security

1. Underlying Host's Linux distribution kernel and its support for hardening, namespaces, cgroups and capability mapping including capabilities and what you do to harden the OS
2. The Container platform layer/engine & APIs: e.g. LXC or Libcontainer
3. Security of the access to the control plane of the host OS, the container engine, scheduler(s) and application deployment platform components
4. Security of the process(es) within the Container

Related: The grouping of containers into trust “zones” (logical/physical) and the networking capabilities to do so (i.e. br0 interfaces insufficient)



But First, A Selfie...

Some Important Questions:

1. Do you run a Linux OS in production?
2. Do you allow developers to run processes atop Linux?
3. Do you allow multiple processes per host?
4. Do you allow processes running atop linux to do so as root?
5. Do you know what 'setenforce 1' means and where you would implement it and why?
6. How do you define a security boundary?



If the vessel isn't secure, how can the containers be?



“Containers Don’t Contain” & “Tupperware Don’t Tupper”

- ◆ Run Docker Engine with AppArmor or SELinux atop a GRSEC kernel to provide containment where isolation is appropriately scoped
- ◆ Don’t run untrusted processes with root privileges & enable user namespaces
- ◆ Map groups of mutually-trusted containers to separate machines; and
- ◆ If you really, really care about isolation and reducing attack surface, run containers in VMs or one container per machine...



The Golden Rule of Containers

If your security sucks now, you will be pleasantly surprised by the lack of change when you move to Containers



RSA®Conference2015

San Francisco | April 20-24 | Moscone Center

Applying This Knowledge



Apply Slide

- ◆ Return to your host-based security fundamentals that you thought you could get rid of because of Cloud...
...which you never did because you still have to armor your guest OS and apps anyway, right? Right!?
- ◆ Leverage DevOps and automation so that security is integrated (Cloud and Containers) into your least common denominator and per-unit of deployment (bare metal, VM, Container, Micro-service...)
- ◆ Don't try this sober.

