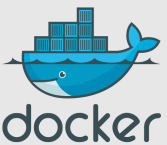




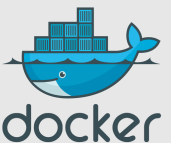
LXC, Docker, Security





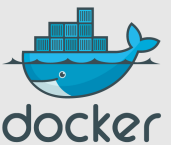
Outline

- Fear, Uncertainty, and Doubt
(and the Awful Truth about LXC and security)
- Some real-world scenarios
(and how to make them safer)
- The road to bullet-proof containers





Fear, Uncertainty, and Doubt





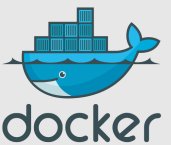
Fear, Uncertainty, and Doubt

“LXC is not yet secure. If I want real security I will use KVM.”

—Dan Berrangé, famous LXC hacker, in 2011.

Still quoted today (and still true in some cases).

But Linux has changed *a tiny little bit* since 2011.





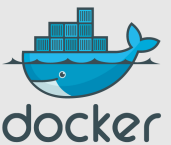
Fear, Uncertainty, and Doubt

“From security point of view lxc is terrible and may not be consider as security solution.”

—someone on Reddit (original spelling and grammar)

Common opinion among security experts and paranoid people.

To be fair, they have to play safe & can't take risks.



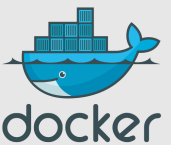


Fear, Uncertainty, and Doubt

“Basically containers are not functional as security containers at present, in that if you have root on a container you have root on the whole box.”

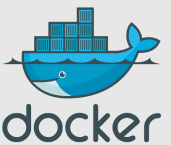
—Gentoo Wiki

That's just plain false, and we'll see why.








The Awful Truth

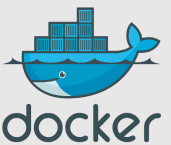




The Awful Truth

Short version:

-  Kernel exploits (e.g. vmsplice exploit)
-  Default LXC settings
-  Containers needing to do funky stuff





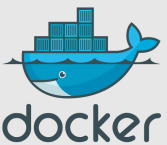
kernel exploits

- You can always do syscalls in a container
- If a syscall is buggy (vmsplice...) and lets you execute arbitrary code, game is over (since it's the same kernel for host and for container)

*However, containers will always (by design) share the same kernel as the host. Therefore, any vulnerabilities in the kernel interface, unless the container is forbidden the use of that interface (i.e. using **seccomp2**) can be exploited by the container to harm the host.*



Ubuntu documentation



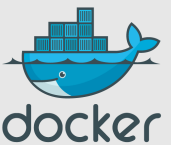


default LXC settings

- If you run containers with all capabilities and permissions, you might as well give full sudoer access to the guest user and complain that “Linux is not secure!”

By default, LXC containers are started under a Apparmor policy to restrict some actions. However, while stronger security is a goal for future releases, in 12.04 LTS the goal of the Apparmor policy is not to stop malicious actions but rather to stop accidental harm of the host by the guest.

—Ubuntu documentation

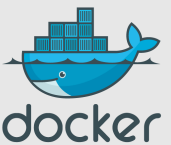




containers needing extra privileges

- Network interfaces (tun/tap...) for VPN or other
- Multicast, broadcast, packet sniffing
- Accessing raw devices (disks, GPU...)
- Mounting stuff (even with FUSE)

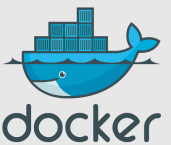
More privileges = greater surface of attack!





Some Real-World Scenarios

(and how to make them safer)



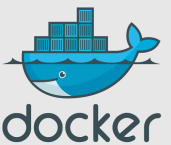


LXC as a payload delivery mechanism (=packaging)



- Without containers, you would run on the host
- ... so don't worry about anything at all!

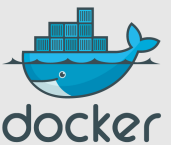
Easy





LXC for development/testing

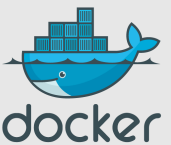
- You run trusted code written by your team (i.e. if someone introduces malicious code, you are in bigger trouble anyway)
- You want to protect against mistakes, not abuse
- LXC will be just fine, especially if you isolate containers on different machines anyway (testing won't hurt production)





LXC for webapps, databases

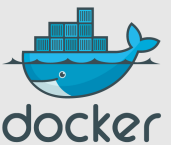
- That stuff shouldn't require root access
 - Run processes as non-privileged user
 - Get rid of SUID binaries (or mount with nosuid)
- You're still vulnerable to buggy syscalls!
 - Keep your kernel up-to-date
 - Or deploy additional layers of security





Reducing syscall attack surface

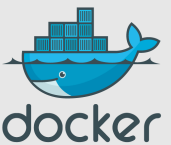
- With seccomp, you can:
 - Limit available syscalls (LXC and Docker: ☒)
 - Limit syscall arguments (not exposed to LXC yet)
 - Switch to a very limited subset (ZeroVM, NaCl)
- Open questions:
 - How much can you drop, and still be useful?
 - How much should you drop, to be secure?





Hardening the L in LXC

- GRSEC helps a lot (not only for LXC workloads)
 - non-executable stack
 - Address Space Layout Randomization
 - protects kernel structures with functions pointers (sets them to be read-only)
 - and many more: check refcount overflows, erase pages (slab, stack...) when the kernel frees them, ...





But I still need/want root!

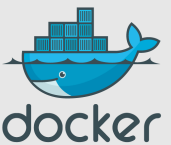
- Use capabilities
 - Bind ports <1024
 - Run tcpdump or other sniffing tools
 - Bypass permission checks (run a fileserver)
 - Lock memory and run quasi-realtime stuff
 - Configure network interfaces, routing tables, netfilter...
 - Renice arbitrary processes
 - etc.





However...

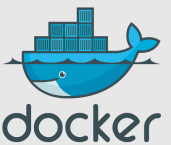
- CAP_SYS_ADMIN is a big can of worms
 - Filesystems (mount, umount, quotactl)
 - Cleanup leftover IPC resources
 - Enter and setup namespaces (setns, clone flags)
 - Many ioctl operations
 - Details like sethostname





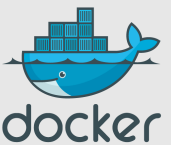
I need to access GPU/raw disk...

- Use the *devices* cgroup controller
 - run X server
 - play OpenGL accelerated games
 - FUN!
 - mine ~~bitcoins litecoins~~ dogecoins
 - PROFIT!
- With Docker, most people just use "-privileged"



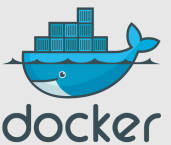


I'm still not convinced.





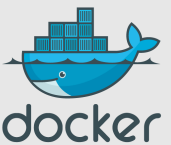
The Road to Bullet-proof Containers





One container per machine

- Use containers for easy and fast deployment
- Use Dockerfiles (or equivalent)
- No overhead at all
 - Disable memory controller
 - Assign macvlan or dedicated network interface
- Retain same workflow in dev and prod!





One VM per container

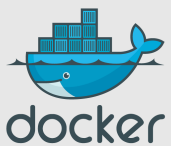
- Run the payload within a VM within a container
- Requires physical machines (or nested VMs)

Plan A (today): magic tricks

- Use 9pfs to hand off the container rootfs to the VM
- Setup transparent network bridge
- Give access to e.g. `/dev/kvm`



Plan B (tomorrow): Docker integration



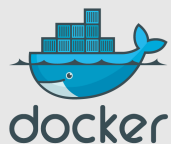


Reminder!

- Security is not a one-shot “let's install this patch, firewall, antivirus, audit mechanism and we're done!”
- Security requires constant updates
(Up-to-date system w/o firewall >> crippled kernel and fw)
- But not all those updates are necessary
(Example: years ago, the Xen/pvgrub exploits)
- Security comes with a cost
(Manpower, but also performance overhead)



→ Know what you really need, and use only that!





Thank you! Questions?

<http://docker.io/>

<http://docker.com/>

<https://github.com/dotcloud/docker>

@docker

@jpetazzo

