Demystifying Container Security

A whirlwind tour of container security features

Aleksa Sarai
Senior Software Engineer
asarai@suse.com

Valentin Rothberg
Container Core Specialist
vrothberg@suse.com

What is a Linux Container?

Introduction to Linux Containers

- **At the core, containers are just processes running on the host**
  - Effective mechanisms to secure, isolate, control and multiplex resources
- **Various use cases ranging from CI to micro-services**
  - Increasing popularity
- **Security skepticism is common and necessary**
  - Our talk demystifies containers security

Namespaces and Control Groups

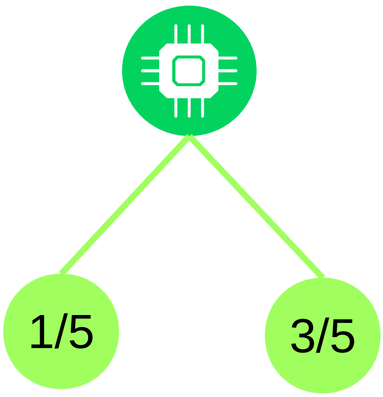
The core of what makes a container a container

Linux Namespaces

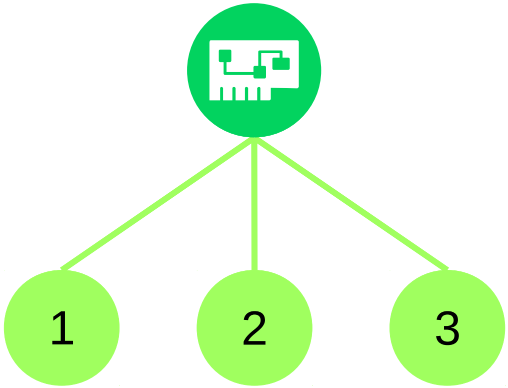
- **Kernel mechanism to isolate a process's view of the system**
  - **Network** devices, stacks, ports, etc.
  - **Mount** points
  - **Process** IDs
  - **User** and **group** IDs
  - **Host-** and **NIS domain** name
  - **Inter-process communication**
  - **Control groups**

Linux Control Groups

- **Control resources for groups of processes**
  - CPU
  - Memory
  - Devices
  - Block I/O
  - Freezer
  - Network priority
  - Processes



CPU Time



Network Priority

Namespaces vs Control Groups

- **The core of what makes a container a container**

- **Very different purposes**
  - Namespaces are for ***isolation*** and ***multiplexing***
  - Control Groups are for ***resource control***

- **More sophisticated versions of older Unix facilities**
  - Namespaces are (sort of) an extension of `chroot`
  - Control Groups are an extension of `ulimit`

Demo

Namespaces and Control Groups

Capabilities

- **A more granular way of providing privileges than just “root”**
  - Examples include mounting, loading kernel modules, network configuration

- **What's the goal of capabilities?**
  - If compromised, the process has limited rights
  - It's safer than running with full-root privileges

- **Open source Docker container engine provides support for giving containers limited capabilities**
  - By default, “root” in a container doesn’t have a full set of capabilities

Demo

Capabilities

SECCOMP

- **Linux technology that allows fine-grained syscall blocking**
  - Block by syscall number, arguments, architecture, eBPF filter, etc.
  - Used in many projects such as Chromium, Firefox, OpenSSH

- **Open source Docker container engine has a default whitelist profile**
  - Support for custom profiles
  - *Note:* Writing your own profile can be quite tedious
  - Default profile has protected against kernel 0-days

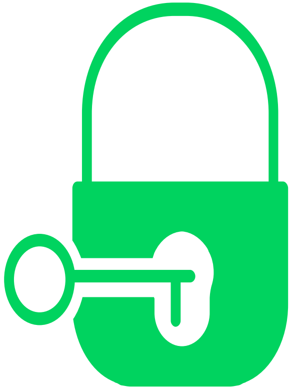
- **Works on any Linux system regardless of Linux Security Module**

Demo

SECCOMP

AppArmor

- **Linux technology that provides path-based access control**
  - Block certain types of access to paths, network resources, capabilities, etc.
  - Provides auditing capabilities
- **Open source Docker container engine has a default profile and supports custom profiles**
  - *Note:* Writing AppArmor profiles can also be tedious
    - See https://github.com/jessfraz/bane and https://github.com/docker-slim/docker-slim
  - AppArmor makes creating profiles based on "normal usage" simple
- **Default Linux Security Module on SUSE Linux Enterprise**
  - Seen as an easier to administer alternative to SELinux



Demo

AppArmor

Runtime Security

- **dockerd still runs as root**
  - Users with access to dockerd have effective root access
  - `rkt` still has a similar problem, despite having better least-privilege principles

- **Open source Docker container engine has support for access control plugins**
  - Very few available – they're hard to implement
  - Custom authentication is not implemented (only client certificates supported)

- **Only give access to dockerd to trusted users!**

Runtime Security

- **Vulnerabilities in a container engine can cause *host* exploits**
  - Docker's default security profiles have protected against Linux kernel 0-days
  - Security is all about layers
    - Requiring a container-engine exploit is another layer
  - User namespaces can help resolve some of the risks

- **Future directions and research:**
  - Rootless containers, a project within the Open Container Initiative
  - Lots of ongoing work to secure kernel primitives used by containers

Best Practices

- **Putting software in a container is safer than on your host**
  - But that doesn’t mean you should abandon common sense

- **Some best practices to get you started:**
  - Don't run as root, and if you do, use capabilities
  - Don’t use your entire host filesystem as a volume, only use what you need
  - Don't make your container a VM, put different services in different containers
  - Don’t run “unconfined”, tailor your security profiles to fit your needs

Image Security

- **Study from 2017 (Shu et al., CODASPY ‘17)**
  - 50% of images have not been updated in 200 days
  - 85% of **latest** official images have been updated in less than 14 days
  - Counter measures are easy!
  - Don’t run untrusted images, audit them like any other software

- **Portus** has integrated image scanning
  - Seamless administration of an on-premise image registry
  - Integrated access control (e.g., LDAP, OAuth2)
  - **zypper-docker** brings our package management to the world of containers
  - CoreOS Clair for additional image scanning

Image Security

- **Open Build Service**
  - Supports building various kinds of images using KIWI
  - Tracks package dependencies of built images
    - Unique feature in the container ecosystem
  - Updating images is easy, secure and automated

Take-Home Message

Running a service inside container is safer than running it outside a container.

SUSE®

We adapt. You succeed.

Unpublished Work of SUSE LLC. All Rights Reserved.

This work is an unpublished work and contains confidential, proprietary and trade secret information of SUSE LLC. Access to this work is restricted to SUSE employees who have a need to know to perform tasks within the scope of their assignments. No part of this work may be practiced, performed, copied, distributed, revised, modified, translated, abridged, condensed, expanded, collected, or adapted without the prior written consent of SUSE.
Any use or exploitation of this work without authorization could subject the perpetrator to criminal and civil liability.

General Disclaimer

This document is not to be construed as a promise by any participating company to develop, deliver, or market a product. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. SUSE makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. The development, release, and timing of features or functionality described for SUSE products remains at the sole discretion of SUSE. Further, SUSE reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All SUSE marks referenced in this presentation are trademarks or registered trademarks of Novell, Inc. in the United States and other countries. All third-party trademarks are the property of their respective owners.