KPLABS Course

Certified Kubernetes Security Specialist

System Hardening

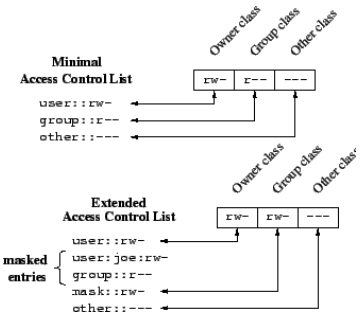
ISSUED BY
Zeal

REPRESENTATIVE
instructors@kplabs.in

Module 1: Overview of AppArmor

1.1 Revising DAC

Discretionary access control (DAC) allows restricting access to objects based on the identity of subjects and/or groups to which they belong.



1.2 Challenges with DAC

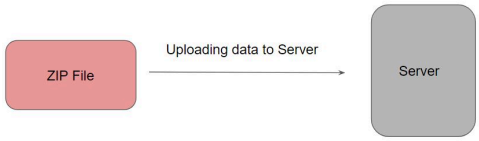
Many of the times, the permissions on the files are more than what is needed.

The programs that are running with the user privilege has complete access to the data owned by that user.

1.3 Sample Use-Case

A client has sent you a zip file on the pretext that it contains certain images related to the error that he is facing while using your program.

In the background, the zip file takes some sensitive data from your host and transmits to the internet.



1.4 Mandatory Access Control

Mandatory Access Control is set by the Administrator.

Two important concepts: Confined (Not Trusted) and Unconfined (Trusted)

1.5 Confined Process

Confined Processes are not trusted.

Everything that the process intends to do must be listed in a profile.

If that capability is not listed in the profile, the process will not be allowed to run that.

1.6 Primary Modes of AppArmor

Following are the two primary modes for AppArmor

Modes	Description
Enforce	Enforces the policy defined in the profile.
Complain	In complain or learning mode, violations of AppArmor profile rules, such as the profiled program accessing files not permitted by the profile, are detected. The violations are permitted, but also logged

Module 2: Integrate AppArmor with K8s

Kubernetes support for Apparmor was added in v1.4

AppArmor profiles are specified per-container.

To specify the AppArmor profile to run a Pod container with, add an annotation to the Pod's metadata:

```
container.apparmor.security.beta.kubernetes.io/nginx: localhost/root.apparmor.myscript
```

It is important to ensure that an apparmor profile is available in the node where the pod will be scheduled.

```
root@ip-172-31-59-221:~# kubectl get pods
NAME             READY   STATUS    RESTARTS   AGE
hello-apparmor   0/1     Blocked   0          24s
```

Module 3: OCI and Container Runtimes

3.1 Importance of Standardization

In an organization, if image standardization is not set, different developers will use a different set of images.

This leads to challenges in troubleshooting, as well as security.

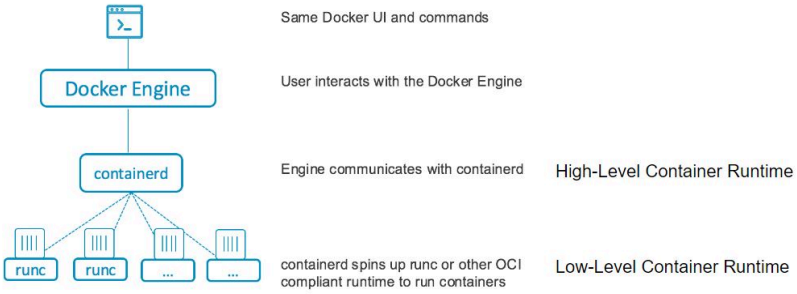
3.2 Open Container Initiative

The Open Container Initiative (OCI) is a Linux Foundation project to design open standards for containers.

There are two important specifications

Specification	Description
Image Specification	Defines how to create an OCI Image, which includes an image manifest, a filesystem (layer) serialization, and an image configuration.
Runtime Specification	defines how to run the OCI image bundle as a container.

3.3 Docker Workflow



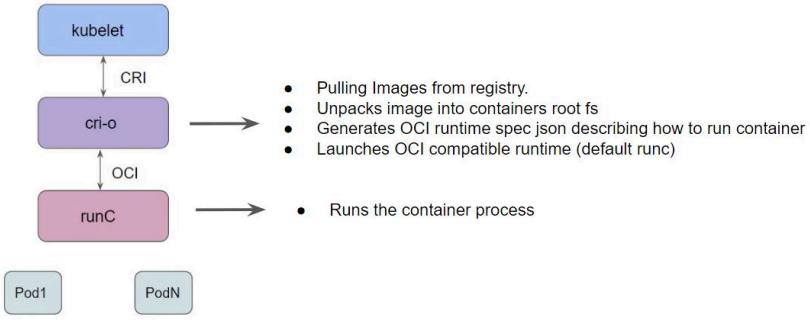
3.4 Container Runtimes

A container runtime is a software that executes containers and manages container images on a node

There are multiple container runtimes available. Some of these include:

- Docker
- containerd
- Cri-o
- Podman

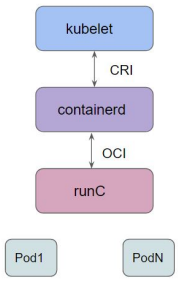
3.5 High-Level and Low-Level Runtimes



Module 4: Container Runtime - Implementation

We will focus on two implementation approaches:

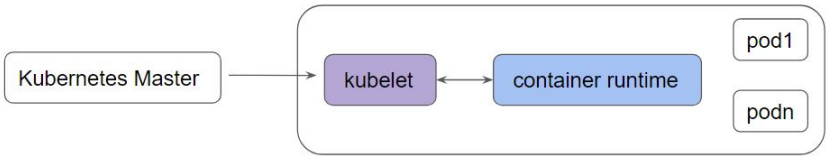
1. Creating containers via containerd
2. Creating containers directly from runC



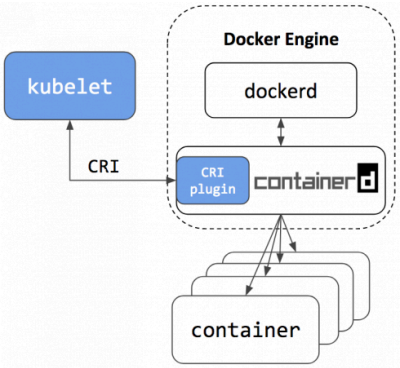
Module 5: Container Runtime Interface

Each container runtime has its own strengths.

K8s uses Container Runtime Interface which is a plugin interface that enables kubelet to use a wide variety of container runtimes without the need to recompile.



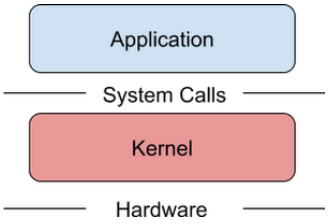
The following diagram shows a sample architecture on how it integrates.



Module 6: Container Sandbox

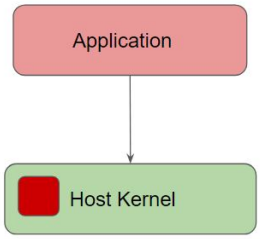
6.1 Basic Architecture

Applications that run in traditional Linux containers access system resources in the same way that regular (non-containerized) applications do: by making system calls directly to the host kernel.



6.2 Use Cases - Bugs

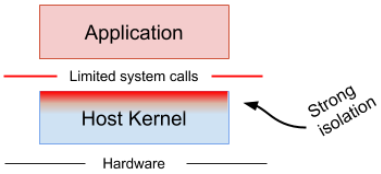
If there are certain bugs at the Kernel level, the application can take advantage of it to achieve various use-cases like privilege escalation and others.



6.3 Seccomp

Kernel features like seccomp filters can provide better isolation between the application and host kernel, but they require the user to create a predefined whitelist of system calls.

In practice, it's often difficult to know which system calls will be required by an application beforehand.



6.4 Container Sandbox

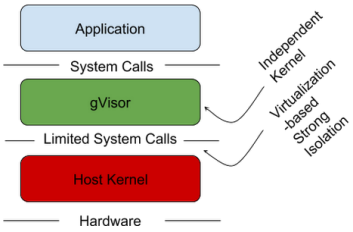
Sandboxing is an approach that enforces a level of isolation between the software running on the machine and the underlying operating system.

In practice, it's often difficult to know which system calls will be required by an application beforehand.

6.5 Overview of gVisor

The core of gVisor is a kernel that runs as a normal, unprivileged process that supports most Linux system calls.

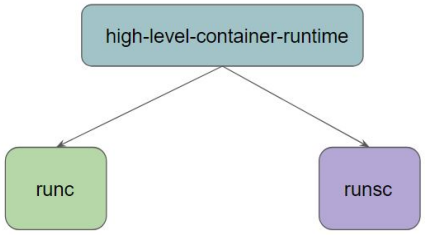
gVisor intercepts all system calls made by the application and does the necessary work to service them thus providing a strong isolation boundary.



6.6 Exploring gVisor

It primarily replaces runc (default runtime) which had few serious vulnerabilities

It comes with an OCI runtime named runsc and hence can act as a drop-in replacement to the runc.



6.7 Exploring dmesg

dmesg (diagnostic message) is a command on most Unix-like operating systems that prints the message buffer of the kernel.

```
root@nginx:/# dmesg
[    0.000000] Starting gVisor...
[    0.485125] Moving files to filing cabinet...
[    0.886586] Feeding the init monster...
[    1.125605] Daemonizing children...
[    1.327612] Verifying that no non-zero bytes made
[    1.395538] Searching for socket adapter...
[    1.737309] Synthesizing system calls...
[    2.108302] Waiting for children...
[    2.290812] Adversarially training Redcode AI...
[    2.402811] Creating process schedule...
[    2.773946] Creating cloned children...
[    3.250078] Ready!
```

gVisor based POD

```
root@nginx2:/# dmesg
[    0.000000] Linux version 5.4.0-1029-aws (buildd@lcy01-amd64
04)) #30-Ubuntu SMP Tue Oct 20 10:06:38 UTC 2020 (Ubuntu 5.4.0-
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.4.0-102
ole=ttyS0 nvme_core.io_timeout=4294967295 panic=-1
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
[    0.000000]   AMD AuthenticAMD
[    0.000000]   Hygon HygonGenuine
[    0.000000]   Centaur CentaurHauls
[    0.000000]   zhaoxin   Shanghai
[    0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 fl
[    0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE re
[    0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX re
[    0.000000] x86/fpu: xstate_offset[2]:  576, xstate_sizes[2]
[    0.000000] x86/fpu: Enabled xstate features 0x7, context si
[    0.000000] BIOS-provided physical RAM map:
```

Default runtime class pod

6.8 Challenges

Generally, an organization makes use of sandboxes like gVisor for the applications that are not entirely trusted (cloning repo from GitHub and running that application)

It can lead to certain performance degradation.

Module 7: RunTimeClass

RuntimeClass is a feature for selecting the container runtime configuration.

You can set a different RuntimeClass between different Pods to provide a balance of performance versus security.


```
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: gvisor
handler: runsc
```

Module 8: Kubeadm with Calico

8.1 Overview of Flannel

Flannel is one of the projects that is developed by CoreOS and is one of the most straightforward and popular CNI plugins.

We have been using Flannel for most of our demos and use-cases.

8.2 Understanding Calico

Flannel is generally considered as the default simple choice however Calico is also popular because of its flexibility and performance.

Calico is also known for its various advanced network capabilities and Network Policy is one of the popular capabilities that is sought.

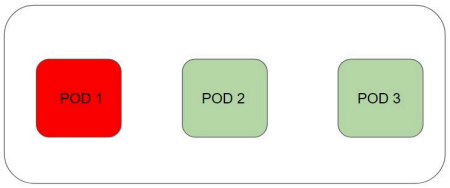
Module 9: Network Policies In-Detail

9.1 Basics of Network Policies

By default, pods are non-isolated; they accept traffic from any source.

A network policy is a specification of how groups of pods are allowed to communicate with each other and other network endpoints.

Example: POD 1 should not have any internal or external communication.



.