Kernel cgroups and namespaces: can they contribute to freedom from interference claims?

Presenters

Priyanka Verma
Senior Software quality Engineer
Functional Safety
In-vehicle OS

Bruce Benson
Sr Automation Engineer (QE)
Functional Safety
In-vehicle OS

Red Hat

Red Hat

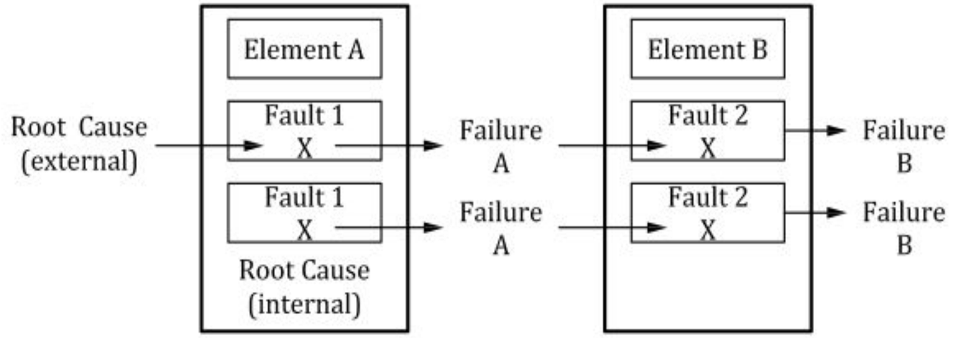
Agenda:

- Why Freedom From Interference (FFI) is important in modern FuSa systems
- Cgroups and Namespaces Overview
- Contributions to FFI claim

Why Freedom From Interference (FFI) is important in modern FuSa systems

What is FFI?
Freedom From Interference is the absence of cascading failures between two or more elements that could lead to the violation of a safety requirement.

It can be described here as: A fault in a less safety-critical (software) component will not lead to a fault in a more safety-critical (software) component.



Why Freedom From Interference (FFI) is important in modern FuSa systems

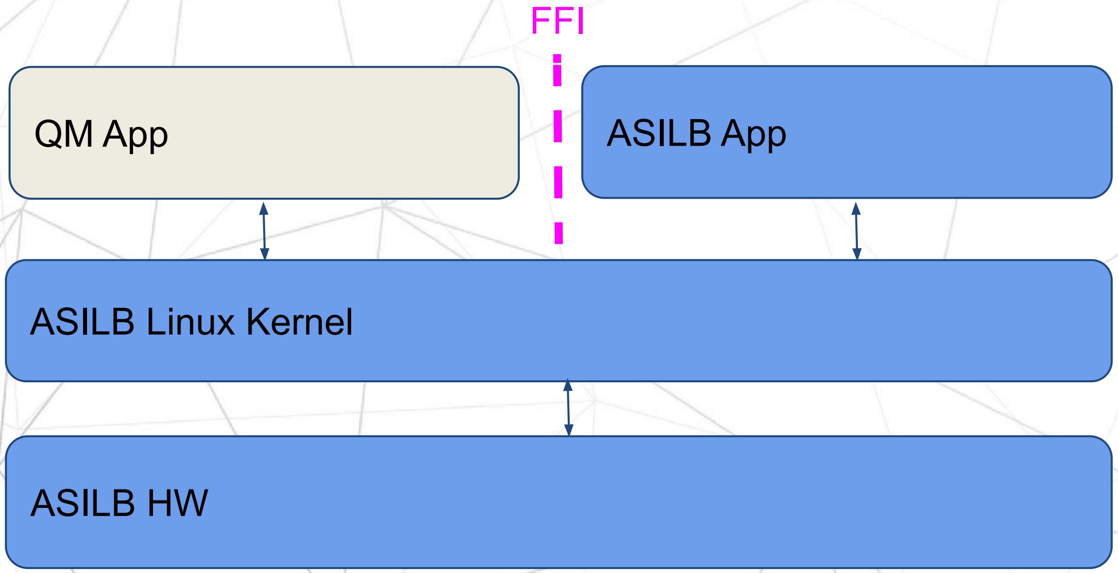
An automotive system consists of multiple software components that interact with each other. The presence of cascading failure from a non-ASIL or a lower ASIL component to a higher ASIL component will lead to one or many safety goal violations.

Types of interference

- Spatial interference - space or memory interference
- Temporal interference - time or CPU interference
- Communication interference - communication channel interference

Why Freedom From Interference (FFI) is important in modern FuSa systems

Automotive system designers are adopting powerful multicore+GPU SoCs that are capable of running diverse workloads concurrently. This prompts thoughts of convergence.



How cgroups and namespaces contribute to the FFI claim?

What are the interference types where these features are more effective?

Cgroup and Namespaces Recap

- All containers running on a host ultimately share the same kernel (somewhere) and system resources.
- A control group (cgroup) is a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, and so on) of a collection of processes.
- Namespaces are then used to limit the visibility of a process into the rest of the system in an exclusive view. This uses the IPC, mnt, net, PID, user, cgroup, time, and UTS namespace subsystems.

LINUX PLUMBERS CONFERENCE September 20-24, 2021

Cgroup

Cgroup provide the following features ("QoS-like things"):

- ❖ Resource limits – a cgroup can be configured to limit how much of a particular resource (memory or CPU, for example) a process can use.
- ❖ Using cgroup not only imposes limitation on CPU and memory usage; it also limits accesses to device files, net, and I/O.
- ❖ Prioritization – how much of a resource (CPU, disk, or network) a process can use can be controlled compared to processes in another cgroup when there is resource contention.
- ❖ Accounting – Resource limits are monitored and reported at the cgroup level.
- ❖ Control – the status can be changed (frozen, stopped, or restarted) for all processes in a cgroup with a single command.

Namespaces

- ❖ Namespaces form a major technology that containers are built on, used to enforce separation of resources and their views (“virt-like things”).
- ❖ Container runtimes like Docker, rkt, CRI-O/podman, LXD, gVisor, Kata, LXC and others make development easier by creating namespaces on behalf of the dev and providing opinionated structure and defaults.
- ❖ There are several different kinds of namespaces supported by Linux:
  - ➢ Unix Timesharing System (UTS)
  - ➢ Process IDs
  - ➢ Mount points (filesystems)
  - ➢ Network
  - ➢ User and group IDs
  - ➢ Inter-process communication (IPC)
  - ➢ Control groups (cgroup)
  - ➢ Time

Contributions to FFI claim

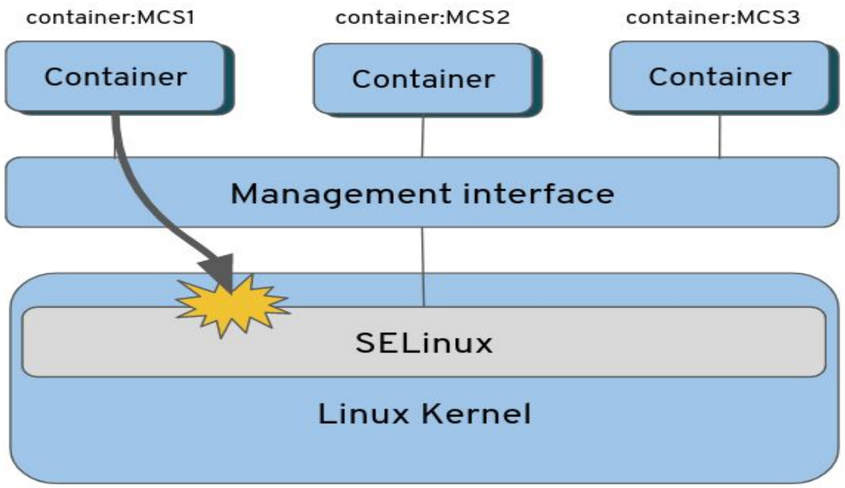
As given in ISO 26262 Part-6, Annex D.2.1: “*The effects of the exemplary faults, and the propagation of the possible resulting failures can be considered.*”

To create isolation, a container engine mainly relies on these two Linux kernel features: namespace and cgroups.

To give the container its own view of the system (isolated from other resources), a namespace is created for each of the resources and unshared from the remaining system. Control groups (cgroups) are also set and used to monitor and limit system resources like CPU, memory, disk I/O, network, etc.

For further security perspective, the container engine can leverage any OS security isolation techniques like seccomp, caps, and LSM —such as AppArmor/SELinux access control—to isolate faults within containers.

(a vendor-specific strategy) SELinux, especially Multi-Category Security (MCS), is central to our support of container separation. In containers we use SELinux to help prevent container attacks against each other and host system resources.



Major categories of isolation capabilities ...and questions

Time interference ("delaying"):

- Some cores can be made invisible to a specific container. Moreover, cgroups allow limit of CPU and memory usage.
- Can we rely on containers for temporal FFI? Exhaustion? Latency?
- What is the role of containers if we have an external flow control monitor?

Spatial interference ("scribbling"):

- Containers support lightweight spatial isolation by providing each container with its own resources (e.g., CPU pins, memory limits, and network and I/O buckets) along with container-specific namespaces.
- Physical devices can be hidden from certain containers, hence cannot be manipulated regardless.

Communication interference ("jamming"):

- If we don't want containers to communicate with each other at all we can set ICC (inter-container communication) to disabled. A network's configuration cannot be changed after creation. So, enabling ICC on an existing network isn't possible.
- If a communication channel is open between two containers, how do we prevent one container spamming the other?