Red Hat
In-Vehicle
Operating System

Red Hat In-Vehicle Operating System: Mixed criticality

Red Hat In-Vehicle OS is a safety-certified Linux platform for software-defined vehicles, allowing safety-critical and nonsafety applications to run on a single ASIL-B certified kernel.

Isolation is achieved using native Linux features and specialized configurations, eliminating the need for separate guest OSs per workload and reducing startup time and memory footprint compared to VM-based approaches. Its unified kernel and container-based architecture streamlines configuration, maintenance, and lifecycle management.

Red Hat® In-Vehicle Operating System (OS) is a production-grade, safety-certified, open source Linux® platform purpose-built for software-defined vehicles (SDVs). Built on strong foundations of industry-leading Red Hat Enterprise Linux, it combines functional safety certification (ISO 26262:2018 ASIL-B) with automotive-grade optimizations. Red Hat In-Vehicle OS's safety certificate includes a mixed-criticality claim, allowing safety-critical applications (up to ASIL B) and nonsafety applications to share a single, ASIL-certified Linux kernel.

Understanding and managing mixed criticality in automotive systems

Mixed criticality refers to a system's ability to isolate applications with varying safety and performance requirements on a shared hardware platform. This is particularly important in automotive systems, where applications can be broadly categorized as:

- **Safety-critical applications:** These are essential for ensuring vehicle and driver safety, including braking systems, collision avoidance, and advanced driver assistance systems (ADAS).
- **Nonsafety applications:** Focused on convenience and entertainment, including infotainment, navigation, and climate control.

As automotive systems shift toward high-performance computing (HPC) platforms, mixed criticality becomes foundational. Key challenges include:

- **Ensuring isolation:** Preventing interference between nonsafety and safety-critical applications to maintain system reliability.
- **Efficient resource allocation:** Distributing shared resources to meet all applications' safety and performance needs.
- **Reliability under stress:** Ensuring safety-critical applications operate as expected during system stress or failures.
- **Adaptability:** Supporting updates, scaling, and new features without compromising safety or performance.

Terminology Note: *Unless otherwise specified, the term "system" in this document refers to the computing environment comprising the Red Hat In-Vehicle Operating System (OS), its supporting software components, and the underlying hardware platform. It does not refer to an automotive "system" as defined by ISO 26262—namely, a set of components or subsystems that includes at least a sensor, a controller, and an actuator.*

Limitations of legacy approaches for workload isolation

Virtualization is commonly used for workload isolation, offering strong separation through distinct kernels. However, this approach comes with notable limitations:

- **System complexity:** Multi-VM architectures complicate continuous integration, demanding coordination across diverse teams, tools, and technologies.
- **Limited scalability:** Evolving workloads are harder to accommodate dynamically.
- **Rigid isolation:** Reliance on hardware-based isolation restricts flexibility.

These challenges necessitate more efficient, scalable, and adaptable solutions.

Red Hat In-Vehicle Operating System certified for mixed-criticality

The Red Hat In-Vehicle Operating System, built on open source Linux, has been certified to meet the requirements of ISO 26262:2018 Parts 3 and 6. This certification validates Red Hat's ability to isolate mixed-criticality workloads, supporting the coexistence of safety-critical applications (up to ASIL B) and nonsafety applications on shared hardware.

How Red Hat In-Vehicle Operating System differs from legacy approaches

Red Hat In-Vehicle OS provides an innovative alternative to traditional VM-based solutions by using a unified Linux-based host with containerization technologies and advanced isolation mechanisms. Containers are lightweight environments that allow applications to run with isolated access to resources such as CPU, memory, and file systems. Key advantages of the approach offered by Red Hat In-Vehicle OS include:

- **Lower resource overhead:** Containers eliminate the need for separate host operating systems, providing lightweight and efficient resource utilization.
- **Simplified management:** A unified kernel architecture reduces complexity, streamlining configuration, maintenance, and scaling.
- **Enhanced scalability and adaptability:** Containers support dynamic workload scaling and modular deployment. Fine-grained software updates can be delivered with minimal dependencies, allowing faster, safer, and more serviceable system changes, including secure OTA updates.
- **Flexible isolation:** Robust isolation mechanisms ensure the secure coexistence of safety-critical and nonsafety workloads without rigid hardware-based isolation.

Technical foundations of Red Hat In-Vehicle OS for mixed-criticality

Linux has long demonstrated its ability to provide Freedom From Interference (FFI), validated by its use in multitenant data centers, where diverse workloads coexist securely on shared hardware. These capabilities—workload isolation, resource management, and security enforcement—form the foundation for mixed-criticality systems. To extend these foundational capabilities into the automotive domain, Red Hat In-Vehicle OS applies tailored strategies that use Linux's strengths while addressing the unique safety and performance requirements of mixed-criticality environments in automotive systems.

- **Partitioned architecture:** Red Hat In-Vehicle OS employs a partitioned architecture to separate safety-critical (ASIL-B) and nonsafety (QM) workloads on shared hardware. This logical separation uses advanced Linux features dynamically configured during system initialization, ensuring the independent operation of each partition while preventing interference and maintaining system integrity.
- **Assumptions of use:** Red Hat defines Assumptions of Use (AoUs) to specify required conditions for the safe operation of Red Hat In-Vehicle OS in functional safety contexts. AoUs establish configuration expectations and guide customers on the intended use of the platform. As key contributors to Linux, Red Hat's maintainers help define these guidelines to support safe deployments.
- **Red Hat Expertise and Validation:** Red Hat provides expertise in configuring Linux systems for the safety use cases.

By combining its maintainers' expertise, partitioned architecture, and validation tools, Red Hat In-Vehicle OS delivers a reliable and adaptable platform tailored to the needs of mixed-criticality workloads while meeting rigorous functional safety requirements.

Red Hat In-Vehicle OS assumed partitioned architecture for isolation

Red Hat In-Vehicle OS takes advantage of logical partitioning, using advanced Linux-based isolation features to effectively manage and segregate mixed-criticality workloads. This approach ensures safety-critical and nonsafety applications coexist securely on shared hardware.

Core components supporting isolation

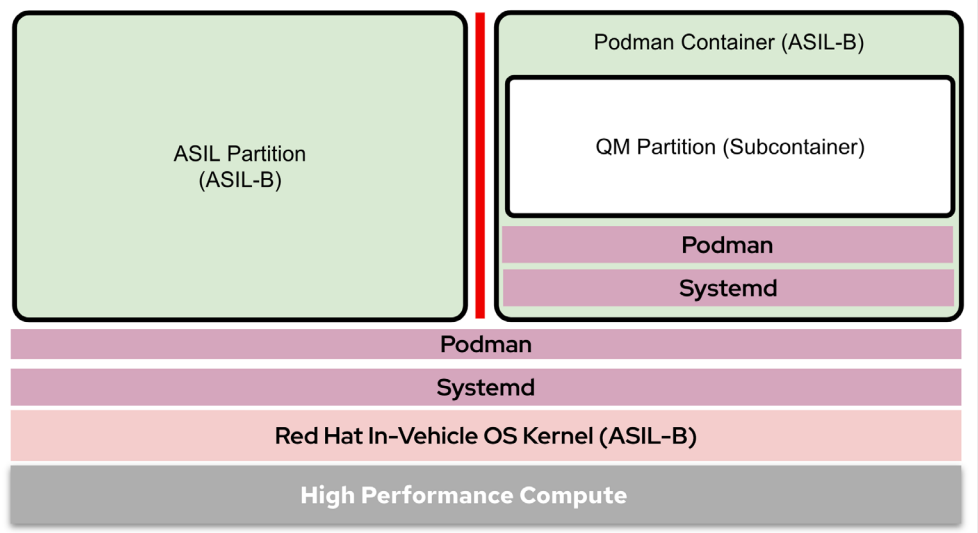
Red Hat In-Vehicle OS achieves the required FFI through key software components:

- **Memory management:** Configures the hardware to establish virtual memory spaces, isolating memory regions between the kernel, containers, and applications to prevent spatial interference.
- **Scheduler:** Manages CPU cycles and enforces time-sharing policies, ensuring predictable execution for safety-critical tasks by preventing temporal interference.
- **Systemd:** Initializes and manages system services, ensuring each application is launched with predefined privileges and isolation mechanisms.
- **Podman:** Configures and manages the QM container, enforcing resource and spatial isolation for nonsafety applications.
- **Glibc APIs:** Provides standard C library calls to support actions such as memory management and dynamic allocation.
- **SELinux and Seccomp:** Implements fine-grained access controls and system call filtering, adding layers of protection against interference.
- **dbus-broker:** Helps secure interprocess communication between services while preventing unauthorized interaction across partitions.

Key elements of partitioned architecture

To ensure FFI between logical partitions, Red Hat In-Vehicle OS employs robust isolation mechanisms through its key architectural components:

- **QM partition:** Hosts nonsafety applications in a logically isolated environment, minimizing latency impacts on safety-critical workloads and supporting deterministic execution.
- **ASIL space:** Safety-critical workloads execute in a noncontainerized user space that operates within the ASIL-B certified environment. Unlike the QM partition, this is not a separate logical partition but a tightly controlled execution context for safety-critical functions.
- **Red Hat In-Vehicle OS kernel (ASIL-B):** The unified Linux kernel provides memory management, process scheduling, and resource allocation while enforcing isolation and efficient resource sharing between safety and nonsafety applications.
- **Hardware platform (ASIL-B):** Includes certified components to ensure safe operation for safety-critical workloads, forming the foundation for robust isolation.



Initialization of system and logical partitions in Red Hat In-Vehicle OS

Red Hat In-Vehicle OS dynamically establishes logical partitions during the startup sequence. This process involves the following key steps.

- **Systemd-based initialization:** During system boot, systemd uses predefined unit files to load and execute essential services. These unit files describe the services to be started, their order, and dependencies, including the initialization of the host environment and the QM partition. If a service crashes, systemd can take actions, such as restarting it, as defined in the unit files.
- **QM partition setup:** The QM partition is dynamically established during the system's initialization process. As part of the startup sequence:
  - **Podman configuration:** The QM partition is implemented as a Podman container, with its execution environment defined by systemd unit files. These files specify the namespaces, cgroups, and SELinux policies that enforce isolation. Namespaces provide process-level

isolation by limiting a process's visibility to specific system resources, such as process IDs, file systems, or network interfaces, while cgroups (Control Groups) manage resource allocation like CPU, memory, and I/O, ensuring that the QM partition operates within its predefined limits without impacting other system components.

- **Independent management:** Within the QM partition, a dedicated instance of systemd manages internal processes independently from the base system. The containerized environment restricts access to the base system resources, ensuring nonsafety workloads within the QM partition remain fully segregated.

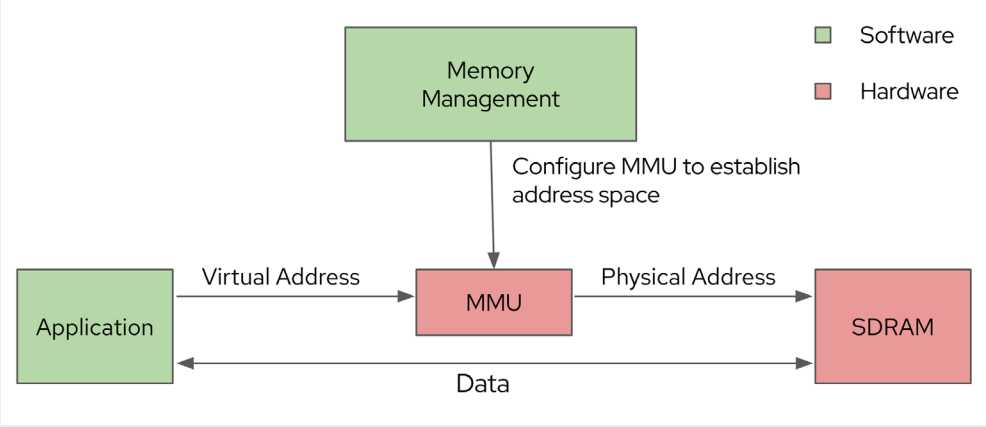
- **Dynamic Enforcement of Isolation:** Kernel isolation features are applied at runtime to enforce strict boundaries between the QM and ASIL-B partitions, ensuring FFI.

Key isolation mechanisms in Red Hat In-Vehicle OS

To ensure FFI between workloads in the QM partition and ASIL space, Red Hat In-Vehicle OS employs the following core isolation mechanisms.

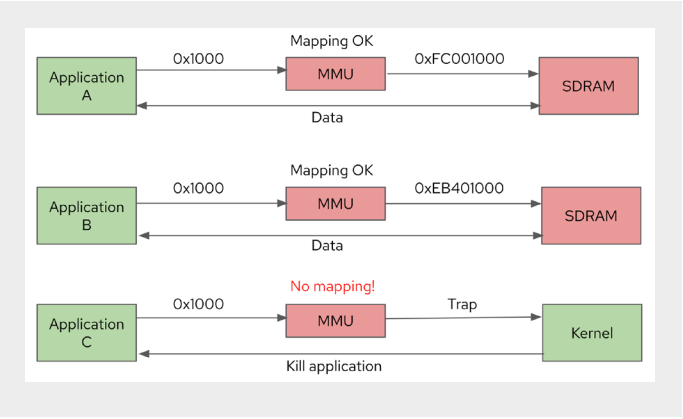
Spatial Isolation

Spatial isolation in Red Hat In-Vehicle OS ensures that applications cannot interfere with one another by isolating their memory regions. The memory management software configures the hardware Memory Management Unit (MMU) to establish unique virtual address spaces for each application.



MMU translates virtual addresses to physical addresses, ensuring isolation for each application

Each application's virtual address space is unique. When an application accesses a valid address within its own space, the MMU translates that virtual address into the corresponding physical location in memory. The MMU generates a page fault if an application attempts to access a memory address outside its defined space. This traps into the kernel, which then terminates the offending application, ensuring that the unauthorized access does not complete. For example:



Applications A and B have overlapping virtual addresses (e.g., 0x1000) but are mapped to different physical memory locations by the MMU.

Application C, which lacks a valid mapping for the same address, triggers a page fault when attempting access and is terminated by the kernel.

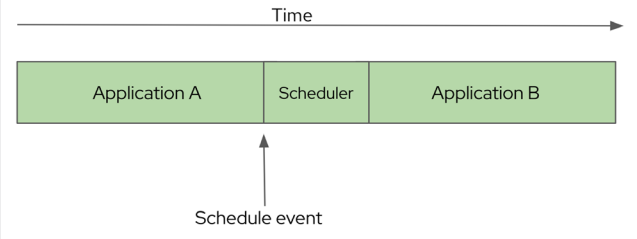
This mechanism not only isolates applications from one another but also ensures they cannot access the kernel's protected memory space. The memory management software provides the foundational spatial isolation required to maintain the integrity of safety-critical and nonsafety workloads in Red Hat In-Vehicle OS.

Temporal isolation

Temporal isolation in Red Hat In-Vehicle OS ensures that safety-critical workloads consistently receive the necessary CPU time while limiting interference from nonsafety-critical tasks. This is achieved through a combination of Linux scheduler configurations, cgroups, and hardware safeguards.

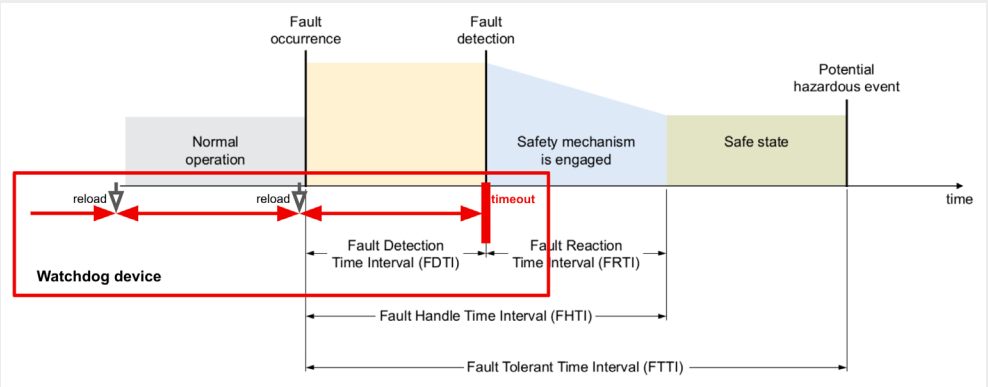
Key mechanisms supporting temporal isolation

- **Linux scheduler with PREEMPT_RT:** The Red Hat In-Vehicle OS kernel, enhanced with the PREEMPT_RT feature set, minimizes interrupt latencies and ensures deterministic scheduling for time-sensitive applications. This feature significantly reduces the time spent in interrupt context, allowing kernel tasks to be preempted by additional interrupts when necessary. By prioritizing interrupt servicing, business-critical tasks can execute without delays caused by long-running interrupt handling, ensuring the system maintains deterministic behavior.
- **Optional CPU resource partitioning with cgroups:** The Linux CPU cgroup feature enforces CPU usage restrictions by grouping processes and allocating predictable resources to safety-critical tasks, ensuring predictable resource allocation. This ensures that nonsafety workloads, such as those running in the QM partition, do not interfere with the execution of time-sensitive, safety-critical operations.
- **Preconfigured QM container for nonsafety workloads:** The preconfigured QM container isolates nonsafety workloads. Using cgroups, this container enforces strict resource limits and boundaries, ensuring that safety-critical workloads retain access to the CPU resources they require. For example:



- Application A is executing when a schedule-related event occurs.
- The scheduler preempts Application A and transitions execution to Application B, which has a higher priority.
- This context-switching mechanism ensures Application B receives the required CPU time without delays caused by lower-priority workloads.

- **Safety watchdog:** As the final safeguard, the hardware watchdog monitors the system for violations of the Fault Detection Time Interval (FDTI). The safety application must periodically reload the watchdog to indicate normal operation. If the safety application fails to reload the watchdog within the specified interval, due to a hang or fault, the watchdog triggers recovery mechanisms to maintain system integrity. Key intervals, such as the Fault Reaction Time Interval (FRTI) and Fault Tolerant Time Interval (FTTI), are critical for detecting and addressing failures promptly, thereby ensuring system safety.



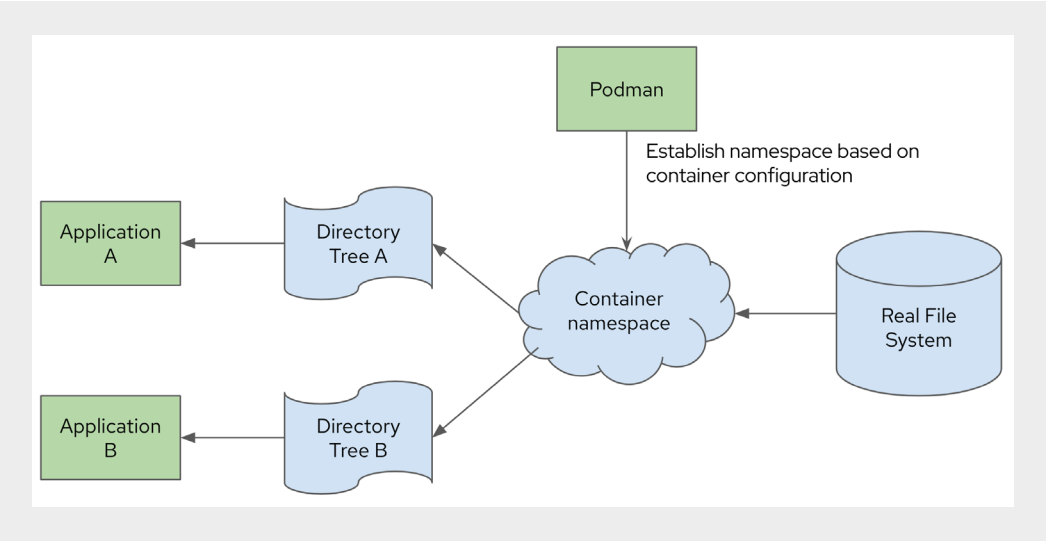
Failure to reload the watchdog on time indicates that a fault occurred

By combining these mechanisms with customer deployment practices that conform to the AoUs, Red Hat In-Vehicle OS guarantees that safety-critical workloads operate as expected, even under heavy system loads or stress. This robust temporal isolation mechanism is critical for maintaining reliability and meeting functional safety requirements in mixed-criticality environments.

Resource isolation

Resource isolation in Red Hat In-Vehicle OS is facilitated by Podman, which helps enforce isolation within the QM partition. Podman establishes namespaces for the QM partition, restricting its visibility and access to system resources. This ensures that nonsafety workloads within the QM partition cannot interfere with safety-critical components or host-level resources.

For example, as shown in the illustration below:



- The **QM partition**, represented as the Podman container, operates within its namespace, providing a controlled and isolated environment.
- The namespace customizes the QM partition's view of the system's file structure and resources, effectively hiding resources like hardware devices, network interfaces, and shared memory segments that are not allocated to the partition.

In Linux, resources such as devices and files are represented as entries in the file system. By restricting the namespace:

- Resources not mapped to the QM partition's namespace are completely invisible to applications within the partition.
- If a hardware device or file is not explicitly allocated, it does not appear in the QM partition's directory tree, making it inaccessible to QM applications.

This layered isolation, combined with other kernel isolation features, ensures that applications within the QM partition remain fully segregated, safeguarding the integrity of safety-critical workloads and maintaining compliance with mixed-criticality requirements.

Kernel address space FFI

Ensuring spatial isolation at the kernel level is a critical challenge in achieving FFI in mixed-criticality systems. In Red Hat In-Vehicle OS, kernel code supporting the mixed-criticality architecture and used by ASIL B applications is ASIL B qualified. To support safety claims, the remaining code is analyzed and verified to ensure it does not interfere with ASIL B functionality.

Unlike user-space isolation, achieving spatial isolation within the kernel requires a multifaceted, layered strategy.

Key strategies for kernel spatial isolation

- **Scope reduction of remaining kernel code:** Unnecessary kernel code is identified and excluded from the Red Hat In-Vehicle OS image, reducing the attack surface and simplifying verification for functional safety.
- **Runtime safety mechanisms:** Red Hat In-Vehicle OS uses kernel-level protections to detect and mitigate issues such as memory corruption, unsafe access, and reference counter overflows or underflows. These mechanisms are validated through targeted fault injection to confirm their effectiveness under stress conditions.
- **Driver certification and risk assessment:** Kernel-mode drivers undergo extensive risk assessments to evaluate their safety and compliance.
  - **In-tree drivers:** These are certified by Red Hat, ensuring adherence to stringent safety standards.
  - **Out-of-tree or loadable modules:** Partners are required to certify these drivers to ASIL B or higher, ensuring that only verified modules can operate within the system.
- **Binary kernel with controlled access:** Red Hat In-Vehicle OS kernel restricts symbol exports, limiting the ability to introduce unvetted modifications. This controlled access prevents developers from inadvertently compromising the kernel's spatial isolation mechanisms.
- **Verification across interference types:** Red Hat supports its functional safety claims for Red Hat In-Vehicle OS through a dedicated verification campaign focused on spatial, temporal, and resource interference.
  - **Spatial interference:** A kernel fuzzing campaign targets all system calls exposed to the nonsafety domain using a KASAN-enabled kernel. The tests must meet predefined safety and reliability criteria.
  - **Temporal interference:** Verified through latency testing on safety-critical workloads while stressing the nonsafety domain with the LTP syscall suite.
  - **Resource interference:** Validated by confirming that resources allocated to safety-critical functions remain isolated when the nonsafety domain is under stress.
- **External watchdog validation:** An external ASIL B hardware watchdog acts as a final safeguard.

By combining these strategies, Red Hat In-Vehicle OS reduces the risks of spatial interference within the kernel, ensuring a stable and security-focused platform for mixed-criticality workloads.

Interprocess protection within a partition

Within Red Hat In-Vehicle OS, Unix-style process protection ensures that individual processes running inside any container, including the QM partition or within the ASIL space, are isolated from one another. For instance, if a QM application spawns multiple processes, kernel-level mechanisms, such as memory access restrictions and process IDs, prevent these processes from interfering with one another.

These protections, while integral to maintaining stability and reliability within a partition, are not the primary focus of this document, as they do not directly contribute to the FFI claims being addressed.

Hardware involvement

The Red Hat In-Vehicle Operating System relies on essential hardware capabilities to enforce isolation mechanisms critical for mixed-criticality workloads. These include:

- **Memory management unit (MMU):** Enforces spatial isolation by applying virtual memory policies configured by the kernel's memory management system.
- **Clocks and timers:** Generate scheduler events required for maintaining temporal isolation.
- **CPU privilege levels:** Protect isolation mechanisms by preventing user-space applications from altering configurations established by the kernel, scheduler, and Podman.

These hardware capabilities are initialized during startup, ensuring the system is fully equipped to enforce isolation before user applications are instantiated.

Failure modes and risk management for mixed-criticality

Red Hat In-Vehicle OS incorporates robust mechanisms to address failure modes associated with mixed-criticality workloads. Potential failure scenarios are analyzed as part of a comprehensive fault tree analysis.

Specific risks associated with the QM partition, such as interference with safety-critical applications, are identified, and mitigation strategies are implemented. These strategies are managed and monitored through Red Hat's safety lifecycle, ensuring compliance with functional safety standards.

For more details, please contact Red Hat at automotive-requests@redhat.com.

facebook.com/redhatinc
@RedHat
linkedin.com/company/red-hat

North America
1 888 REDHAT1

Europe, Middle East, and Africa
00800 7334 2835
europe@redhat.com

Asia Pacific
+65 6490 4200
apac@redhat.com

Latin America
+54 11 4329 7300
info-latam@redhat.com

redhat.com
#2123360-0525