Linux Features for Safety-Critical Systems (LFSCS) WG

Alessandro Carminati - NVIDIA

WG SIG Annual Updates 2026

ELISA

Enabling Linux in Safety Applications

Aerospace · Automotive · Linux Features
OS Engineering Process · Safety Architecture · Systems ·
Tools Lighthouse · Space Grade Linux

What is LFSCS?

Linux Features for Safety-Critical Systems (LFSCS)

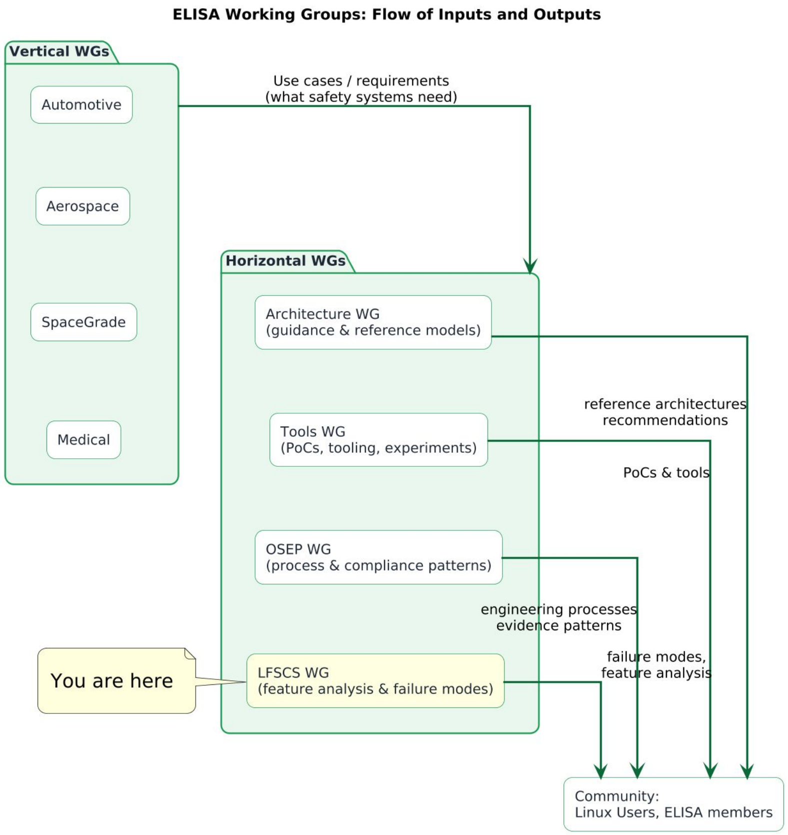
- Working Group within the ELISA Project
- Focused on Linux in safety domains
- Analyzes Linux features in safety contexts
- Identifies potential fault scenarios
- Explores kernel and userspace behaviors

Target application domains

- Automotive
- Aerospace and Space
- Medical Devices
- Industrial and Robotics

Why This WG Exists

- LFSCS is a horizontal Working Group within ELISA
- Works across multiple safety domains
- Complements Architecture, Tools, and OSEP WGs
- Focuses on feature-level and system-level analysis
- Translates domain use cases into technical investigations



Focus Areas and Investigations

- Minimal Linux Footprint
  - Derived from Minimal Config work
  - Identifies essential runtime features
- Memory Isolation and VMAs
  - Core kernel isolation mechanism
  - Complex and failure-prone subsystem
  - Explored lifecycle and allocation behavior
- Members requests
  - Metadata in pointers

Investigations selected to expose potential safety failure modes

Investigation: Minimal Linux

Minimal Feature Artifact

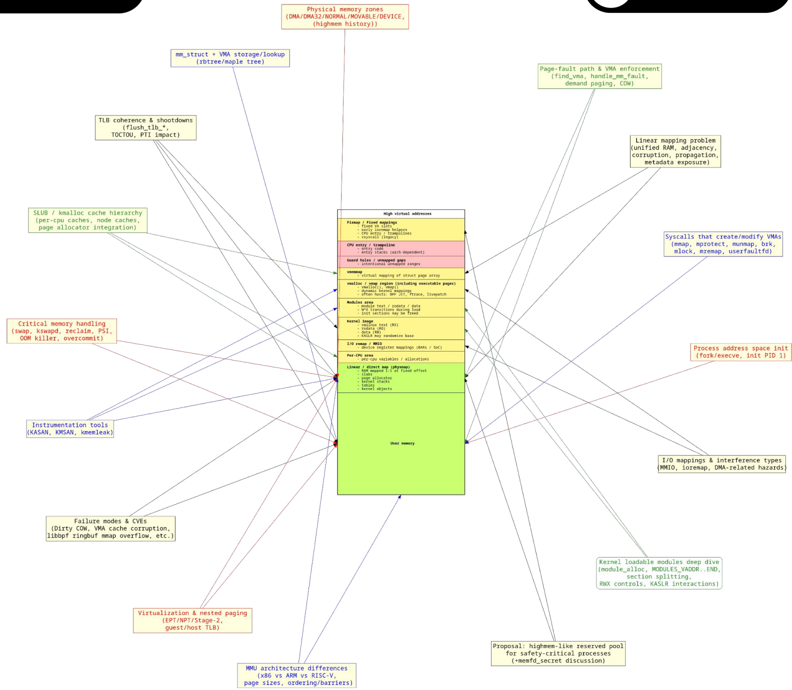
- Motivation
  - Understanding the minimal runtime surface for safety systems
  - Reducing feature exposure and analysis scope
  - Establishing foundational analysis independent of specific use cases
- Approach
  - Builds on prior Minimal Config work
  - Traced real application execution paths
  - Identified essential kernel interactions
- Safety relevance
  - Smaller footprint → fewer failure vectors
  - Clearer feature analyzability
  - Foundations for certification scoping

Investigation: Memory Isolation

VMA presentation

VMA WIP Snapshot

- Motivation
  - ○ Memory isolation underpins mixed-criticality safety systems
  - ○ VMAs form Linux's primary process isolation mechanism
  - ○ Cross-domain relevance independent of specific use cases
- Investigation scope
  - ○ VMA lifecycle and allocation behavior
  - ○ Address space layout dynamics
  - ○ Memory pressure and mapping interactions
- Safety relevance
  - ○ Isolation guarantees vs architectural flexibility
  - ○ Failure propagation across boundaries
  - ○ Foundations for deterministic containment



Physical memory zones
(DMA/DMA32/NORMAL/MOVABLE/DEVICE,
(highmem history))

mm_struct + VMA storage/lookup
(rbtree/maple tree)

Page-fault path & VMA enforcement
(find_vma, handle_mm_fault,
demand paging, COW)

TLB coherence & shootdowns
(flush_tlb_*,
TOCTOU, PTI impact)

Linear mapping problem
(unified RAM, adjacency,
corruption, propagation,
metadata exposure)

SLUB / kmalloc cache hierarchy
(per-cpu caches, node caches,
page allocator integration)

Syscalls that create/modify VMAs
(mmap, mprotect, munmap, brk,
mlock, mremap, userfaultfd)

High virtual addresses

Fixmap / Fixed mappings
- fixed VA slots
- early ioremap helpers
- CPU entry / trampolines
- vsyscall (legacy)

CPU entry / trampoline
- entry code
- entry stacks (arch-dependent)

Guard holes / unmapped gaps
- intentional unmapped ranges

vmemmap
- virtual mapping of struct page array

vmalloc / vmap region (including executable pages)
- vmalloc(), vmap()
- dynamic kernel mappings
- often hosts: BPF JIT, ftrace, livepatch

Modules area
- module text / rodata / data
- W^X transitions during load
- init sections may be freed

Kernel image
- vmlinux text (RX)
- rodata (RO)
- data (RW)
- KASLR may randomize base

I/O remap / MMIO
- device register mappings (BARs / SoC)

Per-CPU area
- per-cpu variables / allocations

Linear / direct map (physmap)
- RAM mapped 1:1 at fixed offset
- slabs
- page allocator
- kernel stacks
- tables
- kernel objects

User memory

Critical memory handling
(swap, kswapd, reclaim, PSI,
OOM killer, overcommit)

Process address space init
(fork/execve, init PID 1)

Instrumentation tools
(KASAN, KMSAN, kmemleak)

I/O mappings & interference types
(MMIO, ioremap, DMA-related hazards)

Failure modes & CVEs
(Dirty COW, VMA cache corruption,
libbpf ringbuf mmap overflow, etc.)

Kernel loadable modules deep dive
(module_alloc, MODULES_VADDR..END,
section splitting,
RWX controls, KASLR interactions)

Virtualization & nested paging
(EPT/NPT/Stage-2,
guest/host TLB)

Proposal: highmem-like reserved pool
for safety-critical processes
(+memfd_secret discussion)

MMU architecture differences
(x86 vs ARM vs RISC-V,
page sizes, ordering/barriers)

Emerging Feedback on Spatial Isolation

- Context
  - Spatial interference through linear mapping was highlighted as an isolation concern during WG investigations and presented to the workshop discussions
  - Early mitigation directions explored approaches leveraging Highmem
- Evolving Landscape
  - Upstream kernel discussions are advancing toward Highmem deprecation and phase-out
- Ecosystem Signal
  - Related mitigation ideas have since surfaced from the broader community
  - Work in this space is expected to be openly published for wider analysis
- WG Posture
  - The WG is tracking these developments where they intersect with ongoing investigations

Side Explorations and Community Input

- Pointer safety models with embedded metadata / extended addressing
- Strong cross-community discussion and visibility
- Engagement with external experts on emerging safety topics

Outputs and Impact

- Investigative Outputs
  - Feature-level safety investigations
  - Failure mode characterization
- Knowledge Artifacts
  - Public technical discussions and documentation
  - Workshop presentations and community sharing
- Cross-WG Collaboration
  - Feeding fault scenarios into tooling exploration
  - Aligning investigations with architectural guidance
- Community Engagement
  - Member-driven topic explorations
  - Open participation and knowledge exchange

Roadmap and Call to Action

- Investigation Continuity
  - Expanding feature-level safety analysis
  - Deep dives into isolation and containment primitives
  - Tracking emerging mitigation approaches intersecting ongoing investigations
- Execution Enablement
  - Establishing collaboration with Tools WG for PoC validation
  - Translating fault analysis into reproducible experiments
  - Collaboration unlocks validation and activates tooling work
- Community Growth
  - Welcoming domain-driven use cases
  - Encouraging member-led explorations
  - Expanding cross-industry participation

Get involved

- Join bi-weekly meetings
  Tuesday @13.00 CE(s)T

- Join the mialinglist

- Join the Elisa Discord

- Participate in the work
  github

Thank you for attending.
Q&A

ELISA
Enabling Linux in
Safety Applications

Aerospace · Automotive · Linux Features
OS Engineering Process · Safety Architecture · Systems ·

Tools Lighthouse · Space Grade Linux