Safety Architecture Working Group update

Gabriele Paoloni, Red Hat

Aerospace · Automotive · Linux Features
Medical Devices · OS Engineering Process
Safety Architecture · Systems · Tools

Topics

- Working group goal & introduction
- Milestones & achievements in 2023
- Challenges and fails
- Plans for 2024

ELISA
ENABLING LINUX IN SAFETY APPLICATIONS

Aerospace · Automotive · Linux Features · Medical Devices
OS Engineering Process · Safety Architecture · Systems · Tools

Working group goal & introduction

Working group goal

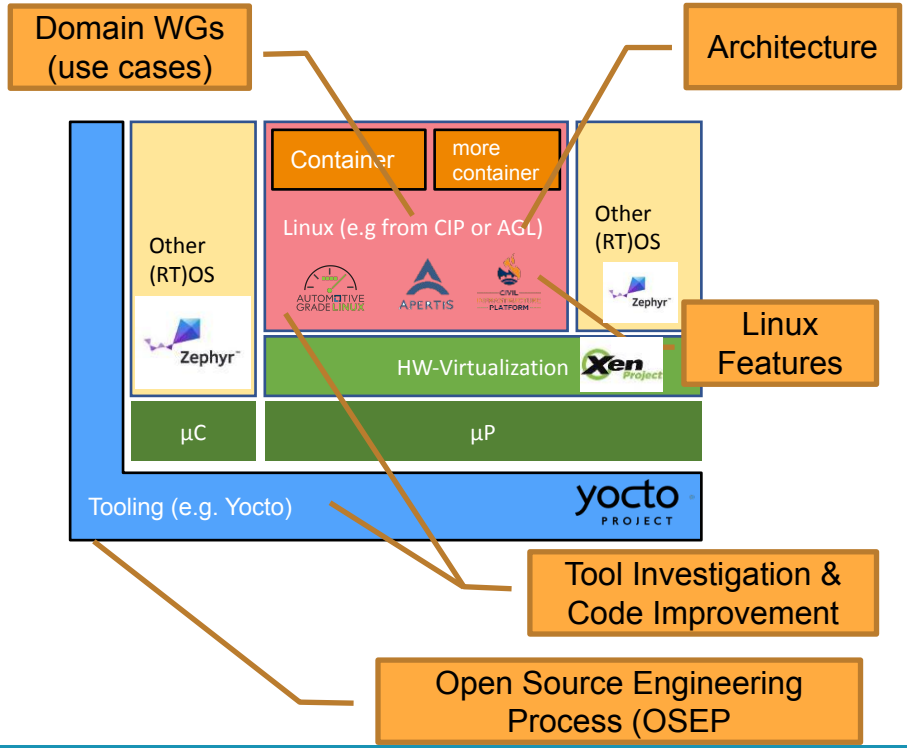
"Determine critical Linux subsystems and components in supporting safety functions, define associated safety requirements & architectural assumptions to deliver analyses for safety critical system integration."

Activities:

- Definition and analysis of Kernel requirements derived from domain WGs (currently the telltale use case)
- Safety analyses inside the Kernel
- Tools and techniques to support an architectural description of the Kernel

The Safety Arch WG within the ELISA WGs

- **Medical, Automotive, Aerospace:** domain WGs analize use cases that define safety requirements for the Kernel
- **OSEP WG:** investigates and proposes the best methodology to perform safety analyses and other safety related activities inside the Kernel
- **Tools WG**: maintains and improves Tools used for the Safety Arch WG and other WGs activities
- **Linux Features**: delves into technical topics that are relevant for the Safety Arch WG and other WGs activities



ELISA
ENABLING LINUX IN SAFETY APPLICATIONS

Aerospace · Automotive · Linux Features · Medical Devices
OS Engineering Process · Safety Architecture · Systems · Tools

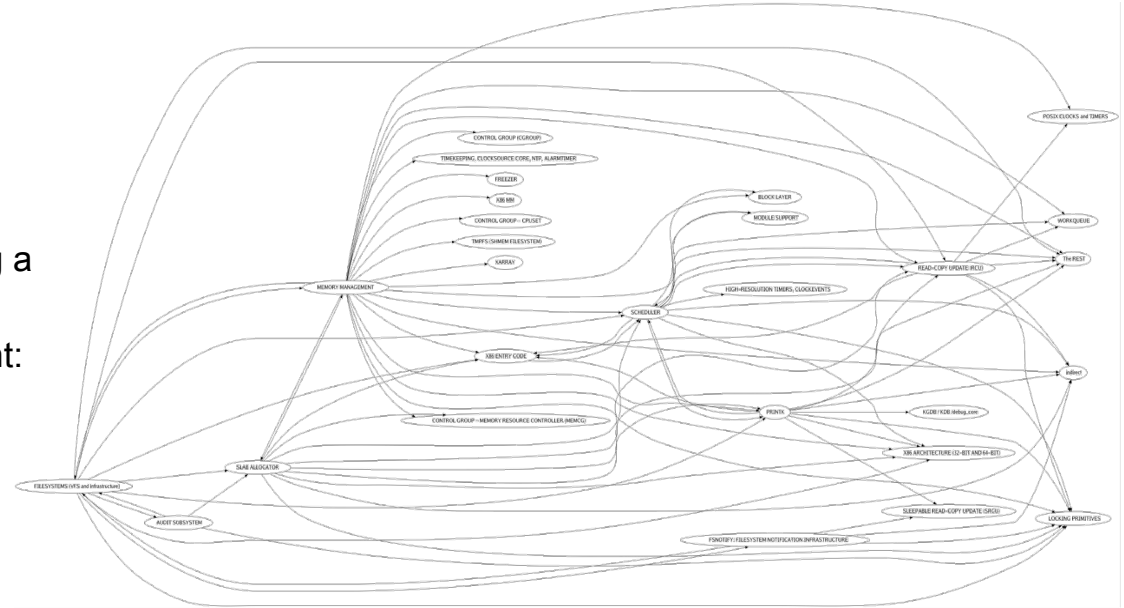
Milestones & achievements

Milestones & achievements

- ks-nav: a tool to provide a static view of the interactions between Kernel components
- STPA(-like) inside the Kernel: investigating the applicability of the STPA methodology to analyse the Kernel against allocated safety constraints, its limitations and proposing improvements accordingly
- Kernel Safety Claims by Runtime Verification Monitors: investigating the value, the methodology and limitations in using RV Monitors to qualify the Kernel against some safety claims

ks-nav Capabilities - Subsystems view

- Visualize subsystem relationships:
  - Illustrate interactions between subsystems in the kernel.
- Understand system structure:
  - Gain insights about relevant subsystems and drivers supporting a specific functionality.
- Impact analysis and change management:
  - Assess changes' effects on subsystems and manage them effectively.
- Safety analysis and fault localization:
  - Identify critical subsystems and support hazard analysis.



ks-nav capabilities - Functions view

Zoom into a single subsystem for…

- Code comprehension and analysis:
  - Visualize the call tree for a specific function.
  - Aid in understanding control flow, debugging, and optimization.
- Safety analysis support:
  - Visualize the impact of compilation settings
  - Exclude code paths that are not relevant
- Impact analysis and change management:
  - Assess the potential impact of code changes and understand propagation effects.

STPA(-like): challenges to apply it to the Kernel

- The Kernel does not come with a “control hierarchy structure”, that is needed to perform STPA. Actually it does not come with SW Architectural Design at all.
- The Kernel is a complex SW component supporting thousands of functionalities over hundred of external interfaces and thousands of internal interfaces. Would the top down hierarchy defined in the STPA work?
- STPA phases are defined in a waterfall fashion and not hierarchically. So all control actions must be defined before defining unsafe control actions (STPA phase 3).

STPA Method Overview

The steps in basic STPA are shown in Figure 2.1 along with a graphical representation of these steps.

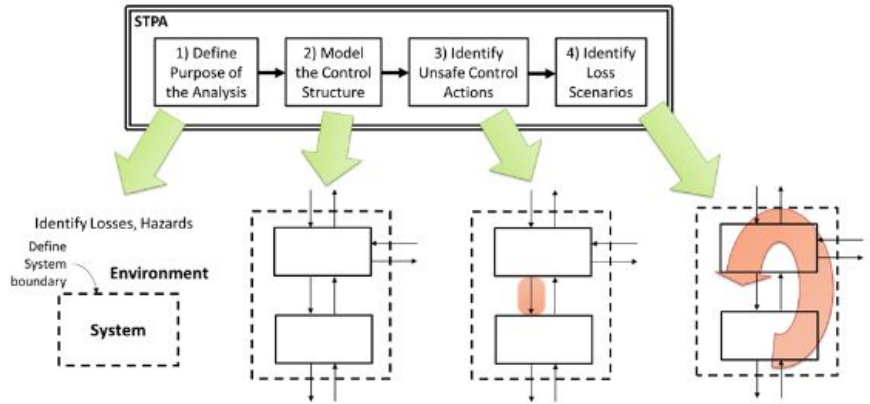


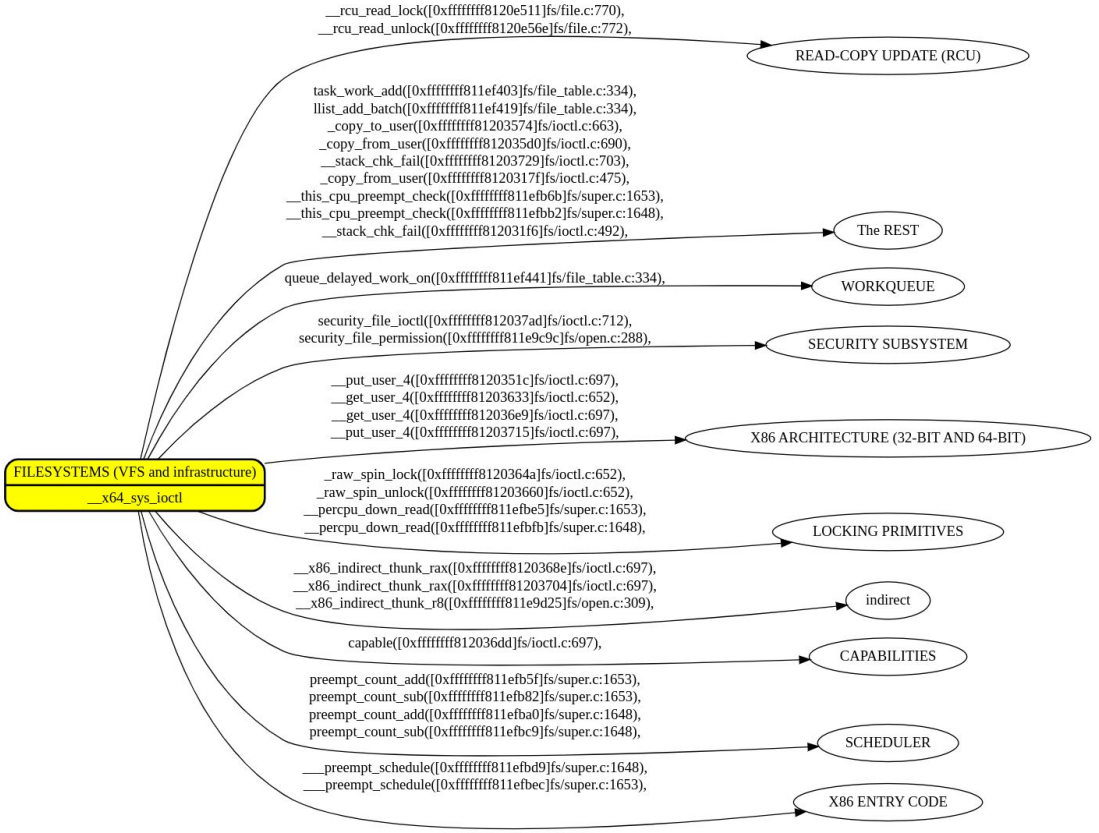
Figure 2.1: Overview of the basic STPA Method

STPA (-like): Kernel Control Hierarchy Structure

The ks-nav tool parses the MAINTAINERS files and the compiled binary Image of the Kernel to determine direct and indirect function calls between subsystems and drivers.

Each interface between subsystems is analysed for possible “control actions”

Control actions here do not follow a waterfall model but a graph model instead

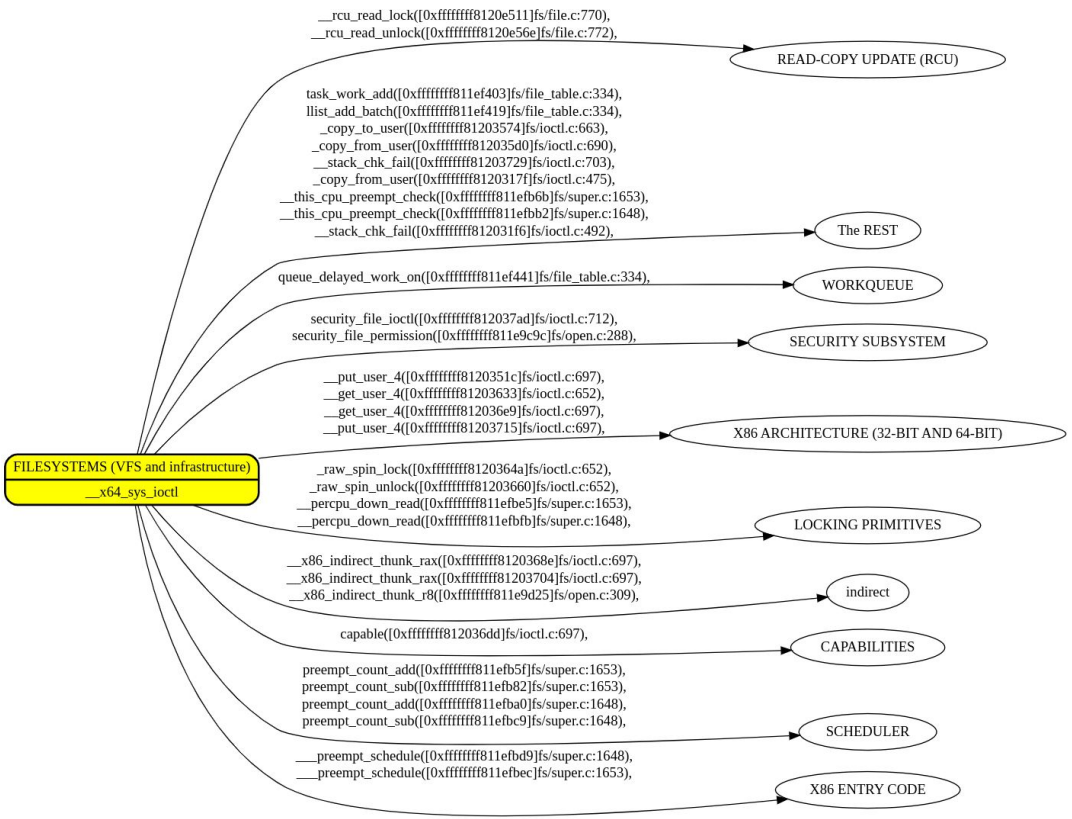


STPA (like): Kernel control actions

STPA phases are defined in a waterfall fashion and not hierarchically. So all control actions must be defined before defining unsafe control actions (STPA phase 3).

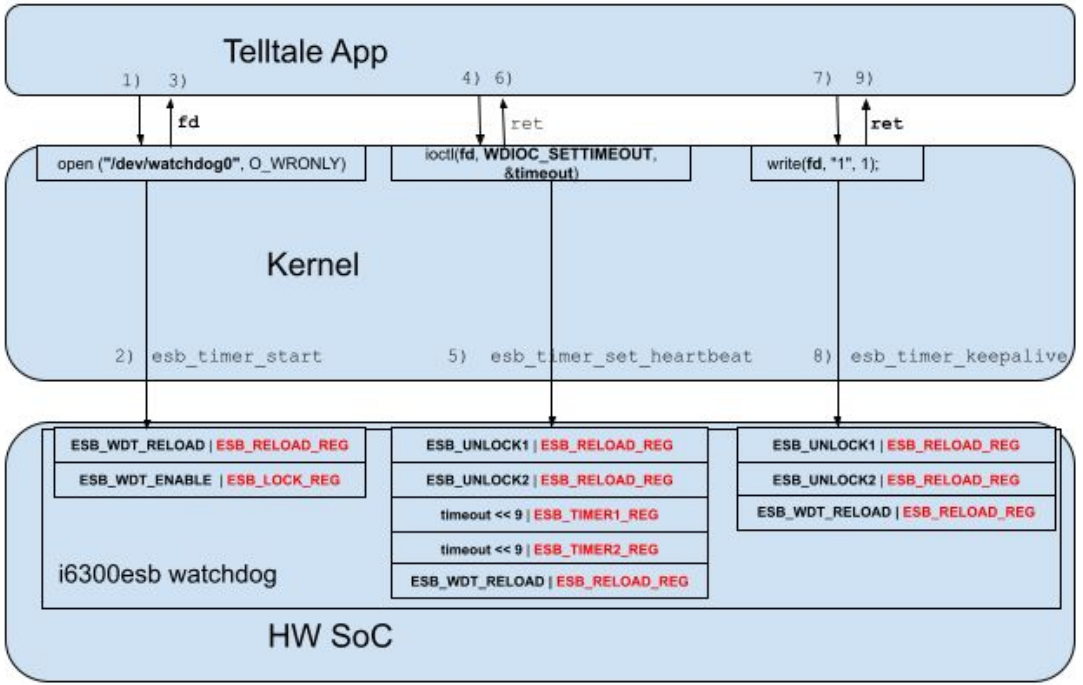
In order to optimize the analysis we completed all STPA phases for each driver or subsystem encountered. So for example in the ioctl() scenario we started from the VFS subsystem (the first one “touched” by the input syscall).

This makes the diagram readable and avoid expanding control actions that have no associated losses/hazards



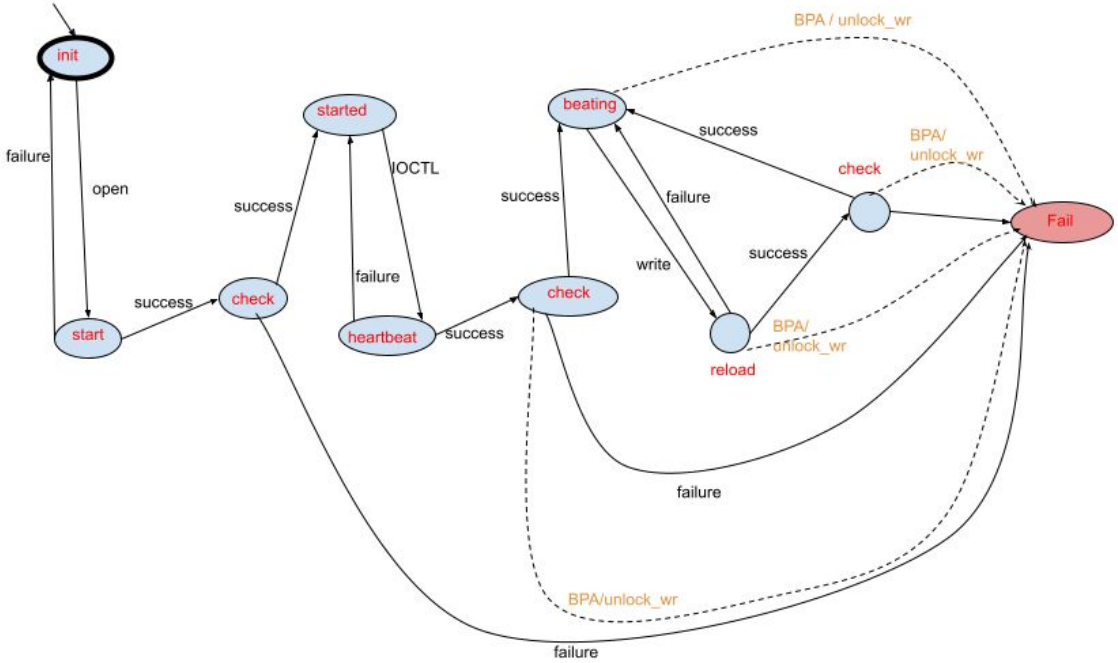
RV Monitors in the Kernel: use case analysis

We considered a use case where the Kernel is used to reliably program an external safety watchdog with a safety timeout and regularly pet such a watchdog



RV Monitors in the Kernel: monitor design

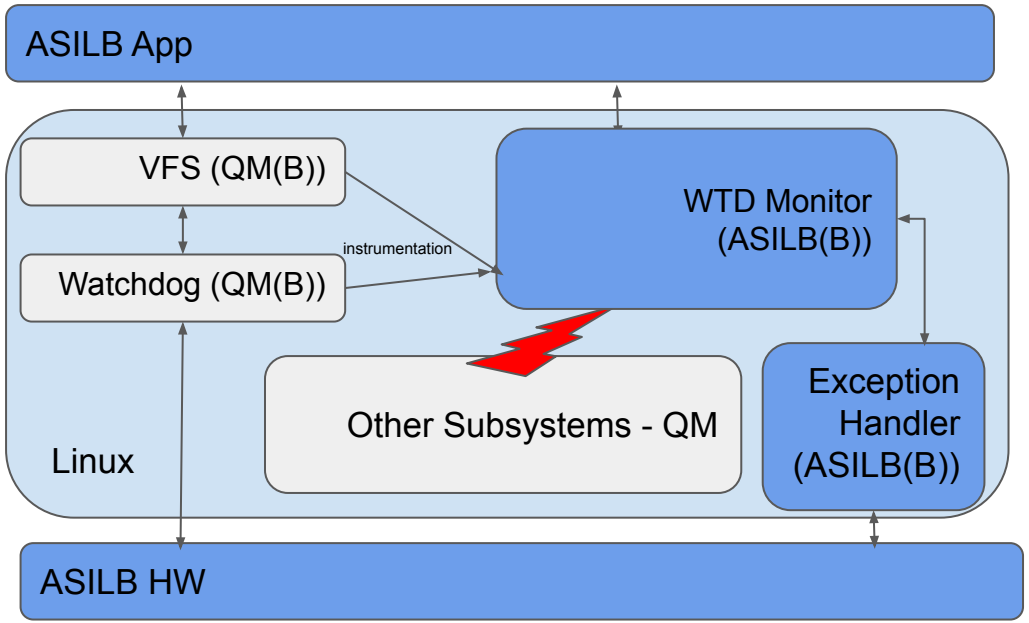
We modelled an RV Monitor to check at runtime the Kernel to behave according to the model



RV Monitors in the Kernel: FFI analysis

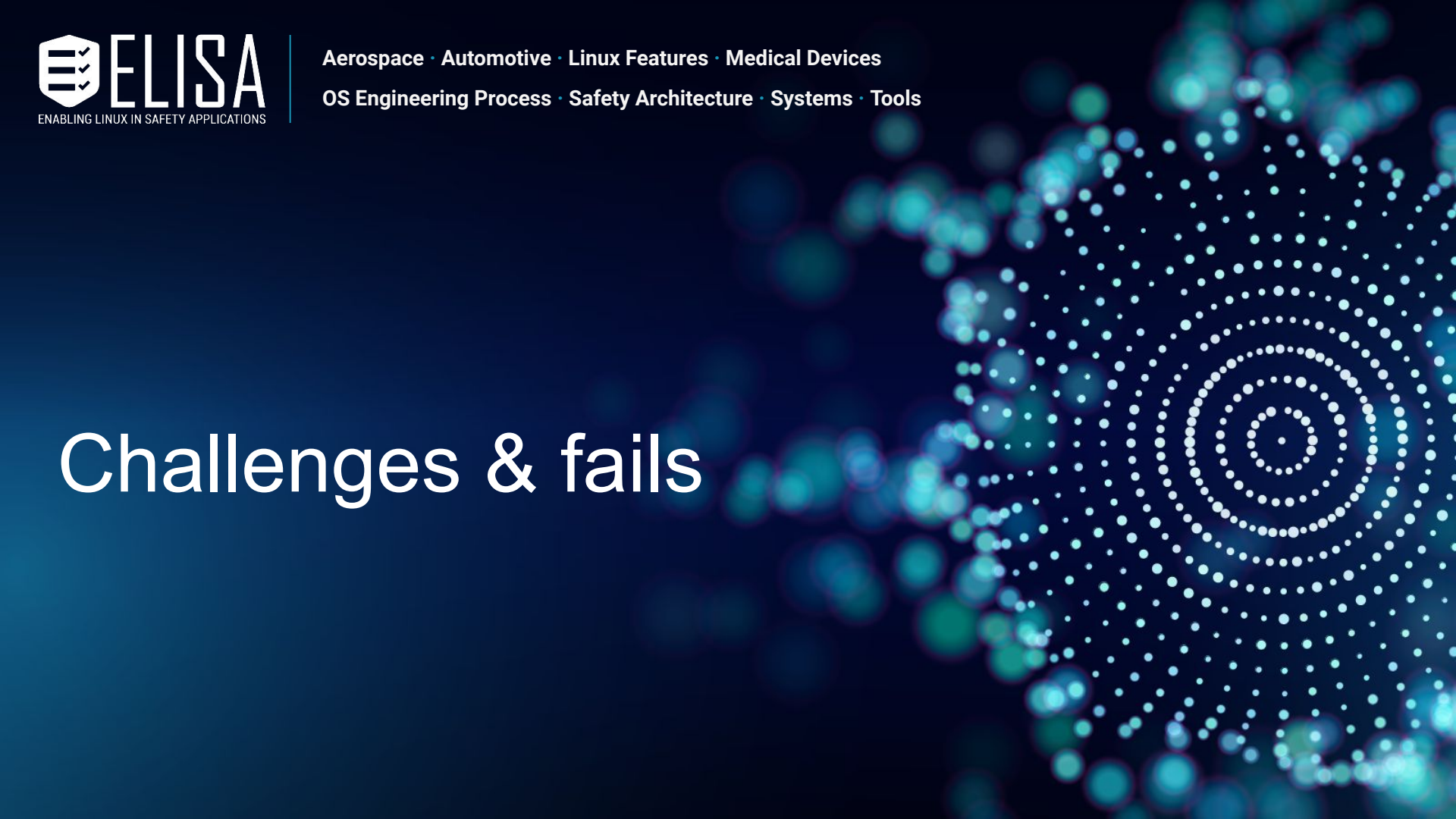
Problem: the RVM lives in the Kernel Address Space. How to protect the RVM from the Kernel itself?

We considered temporal, spatial and communication interferences failure modes and how the RV Monitor addresses them



ELISA
ENABLING LINUX IN SAFETY APPLICATIONS

Aerospace · Automotive · Linux Features · Medical Devices
OS Engineering Process · Safety Architecture · Systems · Tools



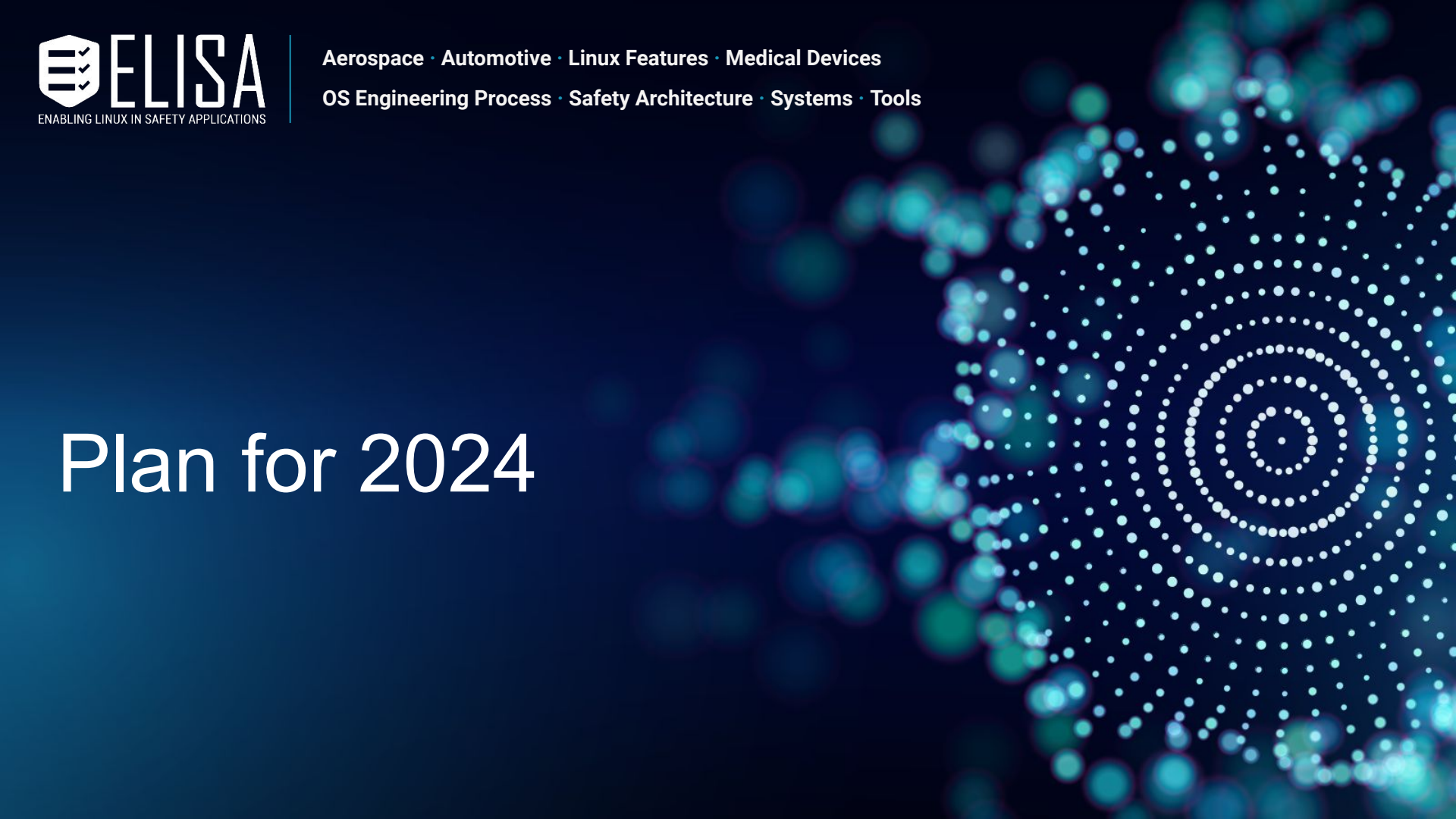
Challenges & fails

Challenges & fails

- STPA: we needed to heavily revisit the STPA methodology to accommodate the Kernel analysis. The WG decided to move away from the STPA in favour of a more flexible hierarchical FMEA
- Lack of Architecture and design documentation for Kernel internals: safety analyses are expensive and time consuming since the Kernel lacks an extensive architecture and design documentation of the code

ELISA
ENABLING LINUX IN SAFETY APPLICATIONS

Aerospace · Automotive · Linux Features · Medical Devices
OS Engineering Process · Safety Architecture · Systems · Tools



Plan for 2024

Plan for 2024

- Develop a methodology to effectively analyse the Kernel (also leveraging expert judgement)
- Apply such methodology to one or more use cases
- Develop an RV Monitor to qualify the Kernel for one or more use cases (analysed above)


ELISA
ENABLING LINUX IN SAFETY APPLICATIONS
Aerospace · Automotive · Linux Features · Medical Devices
OS Engineering Process · Safety Architecture · Systems · Tools
Thank you

ELISA
ENABLING LINUX IN SAFETY APPLICATIONS

Aerospace · Automotive · Linux Features · Medical Devices
OS Engineering Process · Safety Architecture · Systems · Tools

JOIN THE COMMUNITY

ELISA members are defining and maintaining a common set of elements, processes and tools that can be incorporated into specific Linux-based, safety-critical systems amenable to safety certification. ELISA is also working with certification authorities and standardization bodies in multiple industries to establish how Linux can be used as a component in safety-critical systems.
Join us to expand the use of Linux across new industries including healthcare, energy, transportation, and manufacturing. Learn more today to participate and support ELISA.

Join mailing lists | Participate in meetings | Contribute to documentations | Get involved in WGs | Collaborate at Workshops