ELISA
Enabling Linux in Safety Applications

WORKSHOP

NASA Goddard

How to use ks-nav for a feasible and meaningful test campaign in the Kernel

Alessandro Carminati, Luigi Pellecchia

Work in Progress - License: CC-BY-4.0

Agenda

- Introduction to ks-nav
- Testing challenges in the Linux Kernel
- Complexities and challenges
  - Indirect call
  - ftrace
  - High interdependency between kernel functions
- Example of ks-nav usage
- Conclusions

Work in Progress - License: CC-BY-4.0

Introduction to ks-nav

What is ks-nav?

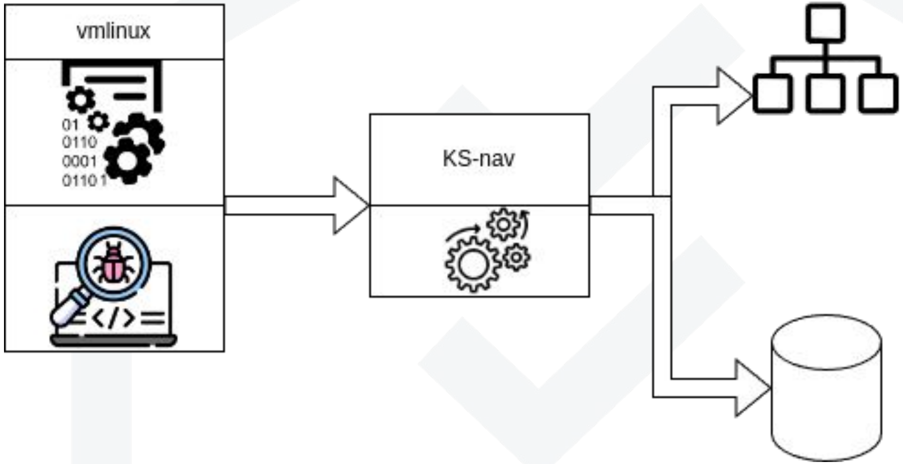
- A toolset that analyzes the Linux kernel binary and produces diagrams to simplify and visualize kernel complexity.

Why Analyze the Binary?

- Avoids challenges of source code analysis (macros, build process, compiler quirks).
- Provides accurate insights directly from the compiled artifact.

Core Features:

- Subsystem-aware diagrams.
- Static call trees.
- Global data usage visualization.

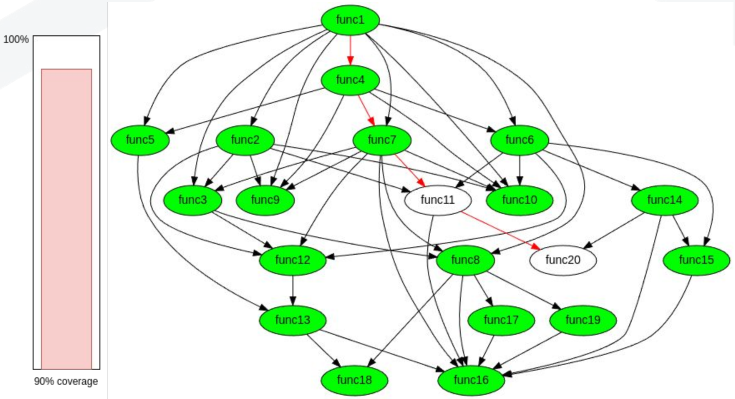


Work in Progress - License: CC-BY-4.0

The Testing Problem

Limitations of Coverage-Based Testing

- **Coverage Without Context**:
  - ➢ Coverage tools indicate which parts of the kernel were executed but lack details about the source.
- **Test Suites and kcov Compatibility:**
  - ➢ Test frameworks like LTP and kselftest were not initially designed to work with kcov, complicating the process of extracting coverage data.
- **Critical Path Uncertainty**:
  - ➢ High code coverage can be misleading, as it may omit safety-critical paths in the context of a specific functionality due to their inclusion while testing other functionalities.
- **Need for Test-Specific Tracing**:
  - ➢ Tools like `ftrace` enable a fine-grained view of test-only paths, allowing precise mapping of code exercised during a specific test campaign.



ELISA
Enabling Linux in Safety Applications

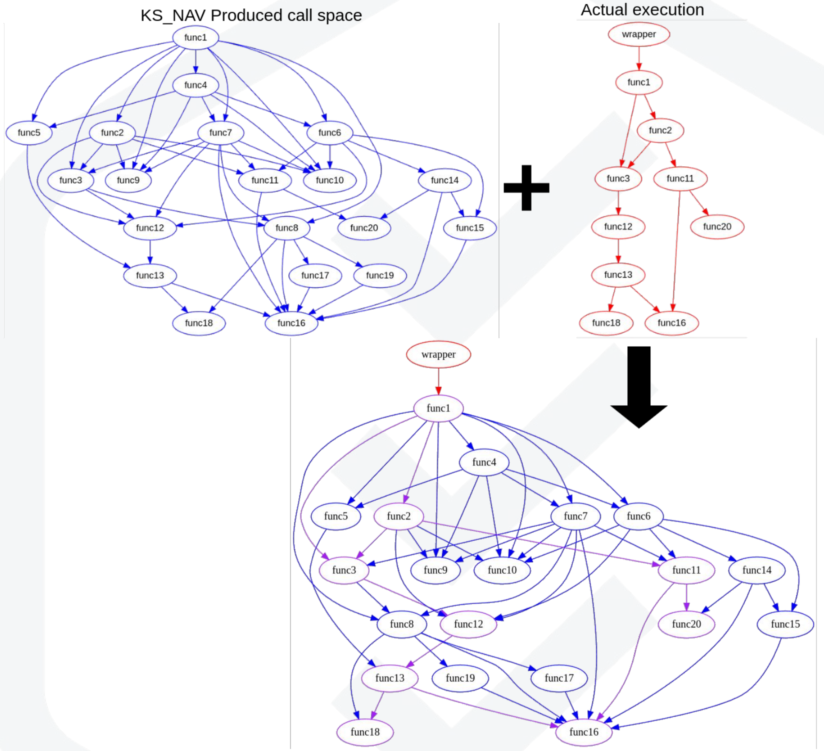
WORKSHOP

Work in Progress - License: CC-BY-4.0

ks-nav Workflow

Highlighting Test Execution vs. Possible Execution Paths

- **Static Call Tree Generation**:
  - ➢ ks-nav performs static analysis, producing the call tree for a given top-level API.
  - ➢ This tree represents all theoretically reachable paths in the code.
- **Analysis of Static Call tree and definition of Critical Path**
  - ➢ Ks-nav provides simplified views of the code.
  - ➢ Expert driven activity aimed to provide a list of critical path that need to be tested.
- **Dynamic Matching with Test Data**:
  - ➢ ftrace collects execution data during tests, creating a runtime call tree.
- **Ensuring Coverage of Critical Paths**:
  - ➢ Runtime tree is matched against the static call tree for granular function-level coverage.
  - ➢ Critical paths identified and explicitly verified against the runtime data.



ELISA
Enabling Linux in Safety Applications

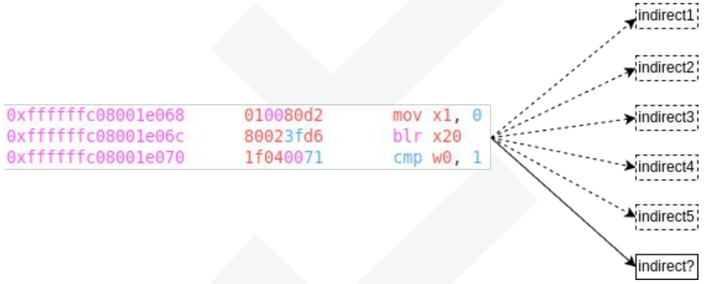
WORKSHOP

Work in Progress - License: CC-BY-4.0

Indirect Calls’ Challenge

Challenge of Analyzing Indirect Calls in ks-nav

- **Nature of Indirect Calls**:
  - ➢ Indirect calls are extensively used in driver-based architectures like the Linux kernel.
  - ➢ These calls delegate execution to functions resolved only at runtime.
  - ➢ Some architectures may implement these calls with additional obfuscation to mitigate vulnerabilities (e.g., Spectre).
- **Binary Analysis Limitation**:
  - ➢ Enumeration of possible targets from the binary image alone is impractical due to complexity.



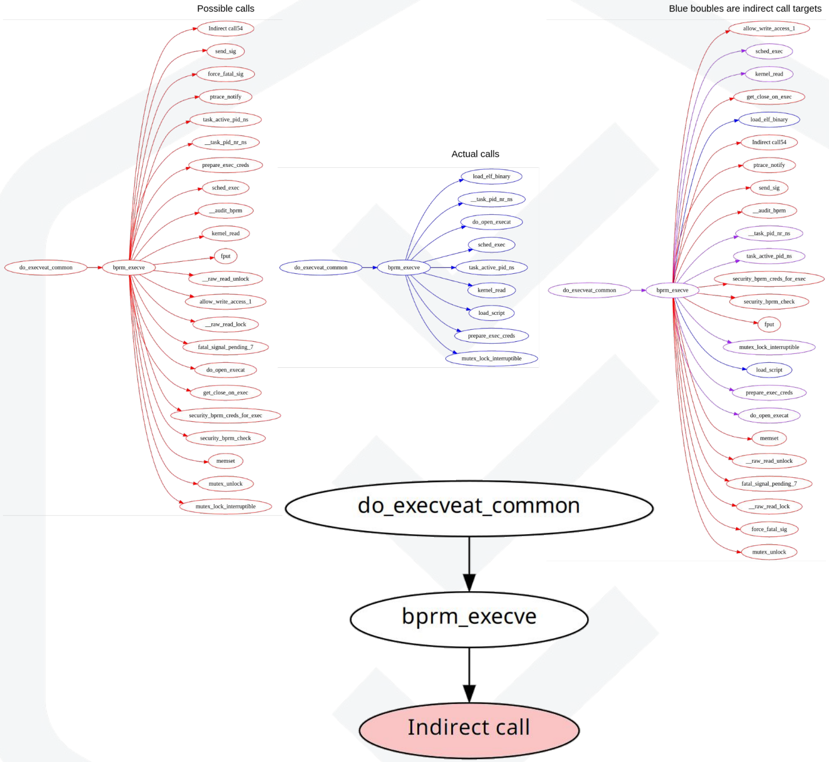
ELISA Enabling Linux in Safety Applications
WORKSHOP

Work in Progress - License: CC-BY-4.0

Indirect Calls' Impact

Effect of Indirect Calls on ks-nav Diagrams

- **Interruption of Call Tree Exploration**:
  - Indirect calls halt the static exploration of code paths.
  - The resulting ks-nav diagram ends at the indirect call, leaving the downstream paths unexplored.
- **Impact on Diagram Completeness**:
  - Critical execution paths may remain unrepresented in the call tree.
  - This incompleteness can obscure potential issues and hinder coverage assessment.



ELISA Enabling Linux in Safety Applications

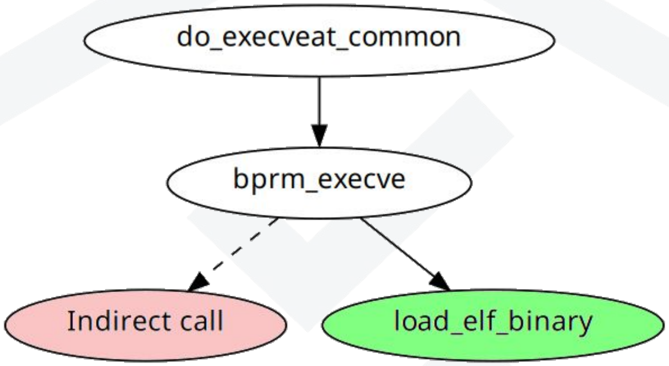
WORKSHOP

Work in Progress - License: CC-BY-4.0

Indirect Calls: a Solution

Indirect Calls on ks-nav possible solution

- **Unique Opportunity in Kernel Context**:
  - Unlike generic software, the kernel includes all possible code paths in a given build.
  - This makes it possible to statically enumerate possible targets, even if the exact runtime call remains unknown.
- **Potential for Recovery via ftrace**:
  - ftrace logs provide the actual runtime resolution of indirect calls.
  - These logs enable amendment of the ks-nav diagram by re-running ks-nav with ftrace-informed APIs.



Work in Progress - License: CC-BY-4.0

ftrace Limitations' challenge

Root Cause Investigation

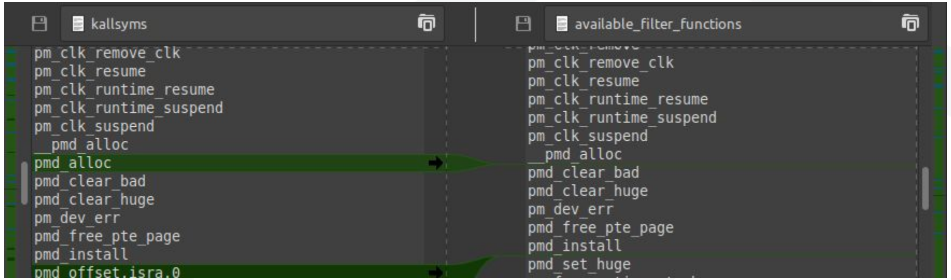
- **Instrumentation Assumption**:
  - ➢ Assumed `available_filter_functions = kallsyms - noinstr`.
  - ➢ Reality: Not all functions in `kallsyms` are included in `available_filter_functions`.
- **Compiler Behavior**:
  - ➢ Functions must be instrumented for ftrace to log them.
  - ➢ Apparently `static` linkage functions can miss instrumentation code due to compiler optimization.
- **Complexity of Mechanism**:
  - ➢ Not a straightforward `not in log` relationship.
  - ➢ Compiler optimizations and directives impact instrumentation unpredictably.

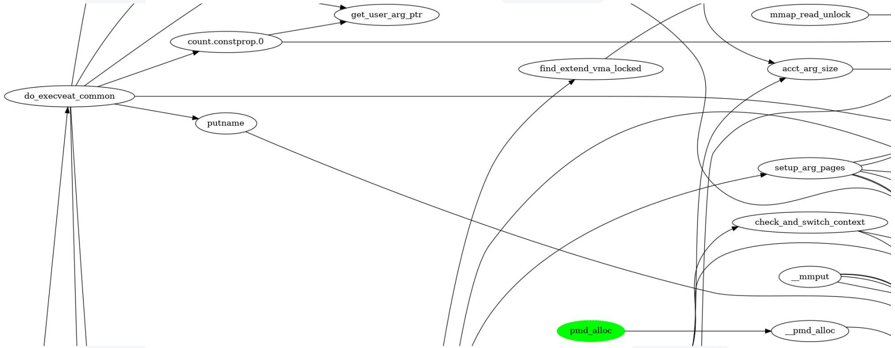
Work in Progress - License: CC-BY-4.0

ftrace Limitations' impact

Impacts on Graph Generation

- **Gap in Instrumentation**:
  - ➢ Some functions are excluded from ftrace logging, leading to incomplete execution data.
- **Graph Inaccuracy**:
  - ➢ Missing log entries cause root nodes for new, disconnected graphs.
  - ➢ Key relationships and execution paths are misrepresented.
- **Impact on the analysis**:
  - ➢ Critical execution paths cannot be fully traced.
  - ➢ Results in unreliable testing completeness assessments.





ELISA
Enabling Linux in Safety Applications

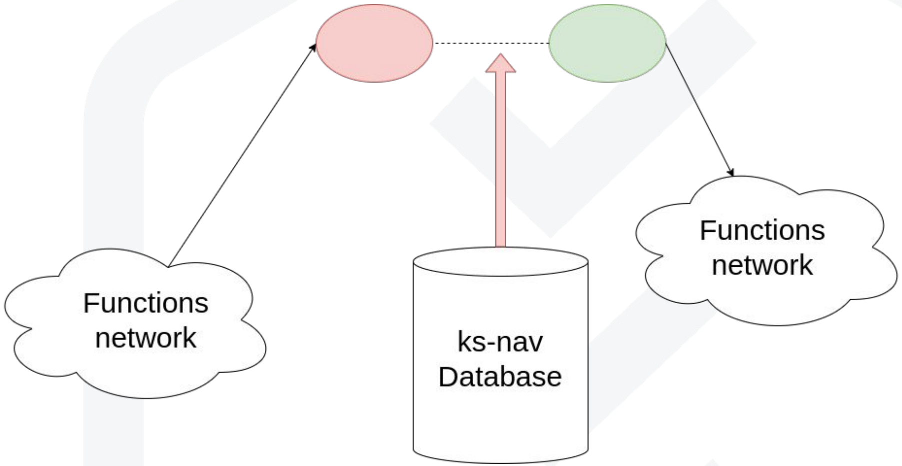
WORKSHOP

Work in Progress - License: CC-BY-4.0

ftrace Limitations: a Solution

What can be done for this?

- Change Compilers flags to make this event more unlikely
- Enhance ftrace produced graphs by integrating information from the ks-nav database in the post processing phase.



ELISA Enabling Linux in Safety Applications

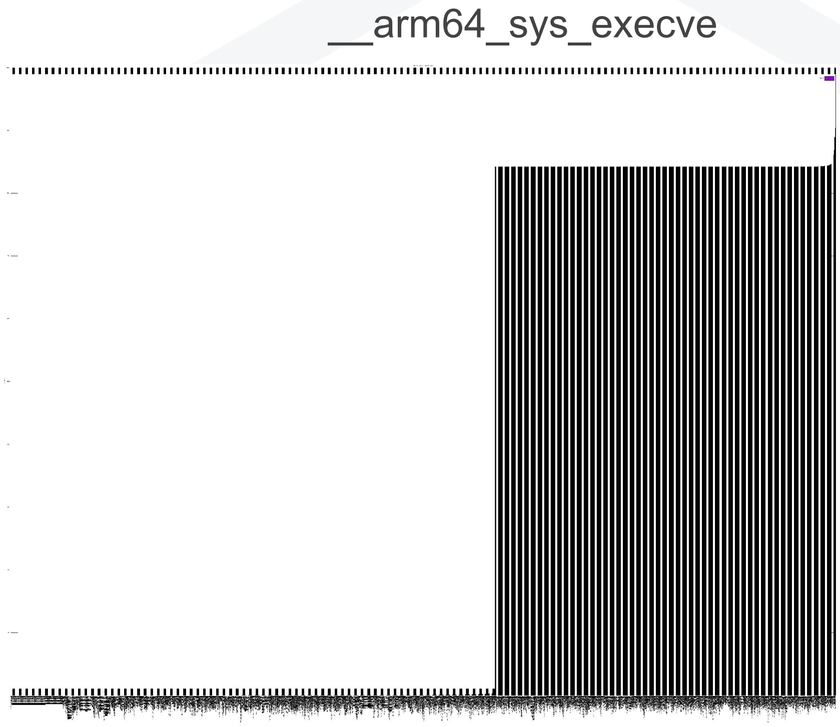
WORKSHOP

Work in Progress - License: CC-BY-4.0

High interdependency

The Challenge of Visualizing Kernel Call Trees

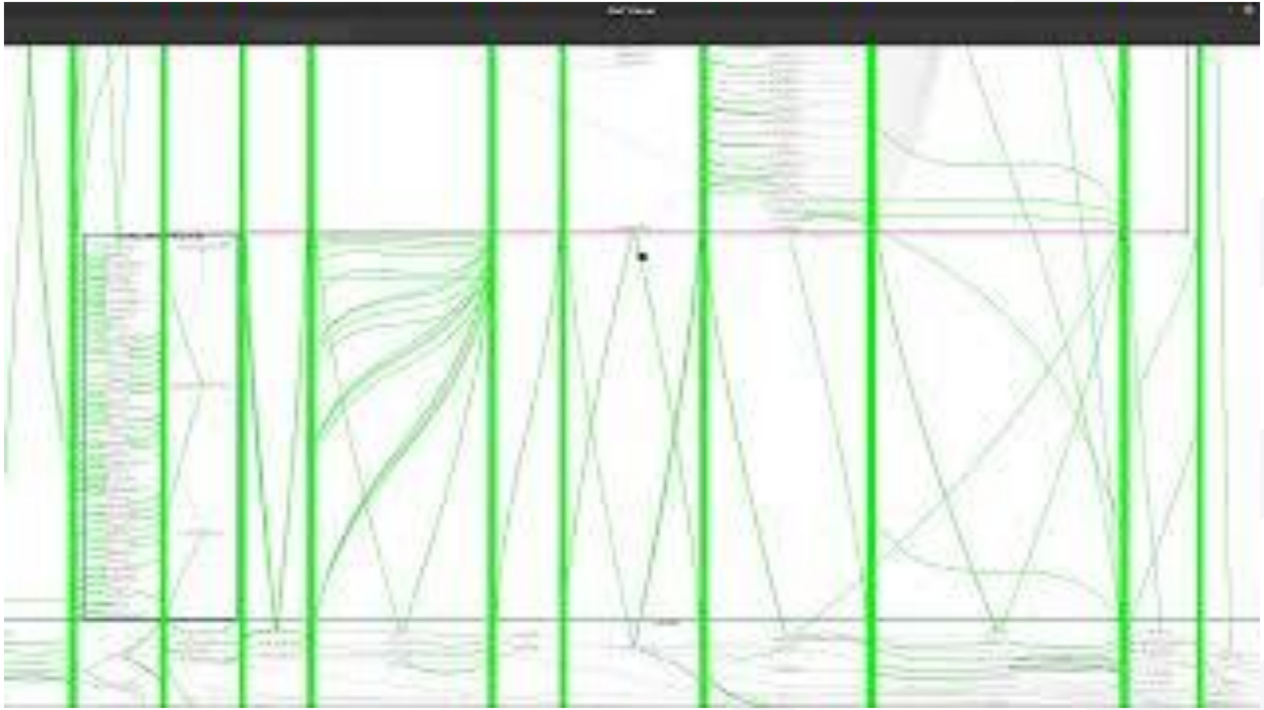
- **Observation:** Analysis of the call tree reveals a significant subset of functions with a high and identical number of reachable arches.
- **Implication:** These functions form a tightly interconnected core, where each function can potentially reach the others.
- **Challenge:** Simplifying such a dense graph by collapsing subgraphs into single nodes is not feasible due to the pervasive interdependencies.
- **Consequence:** Efforts to reduce complexity for visualization and analysis are hindered by this inherent structural characteristic.
- **Solution:** Use subsystem to have the graph partitioned, or use a different strategy, like interrupt graph exploration. `graph_tool` supports both strategies.



ELISA Enabling Linux in Safety Applications WORKSHOP

Work in Progress - License: CC-BY-4.0

Example: ks-nav in Action



ELISA
Enabling Linux in Safety Applications

WORKSHOP

Work in Progress - License: CC-BY-4.0

Future Work

- **Indirect Call Handling in ks-nav**
  - ➢ Current Challenge: Indirect calls interrupt call tree exploration in static analysis.
  - ➢ Planned Approach:
    - Extract indirect call positions from the binary.
    - Use debug info and **libclang** to identify the object type and resolve potential targets from source code.
    - Introduce support for architectures with unique binary-level indirect call mechanisms.
- **ftrace Log Translation Improvements**
  - ➢ Current Challenge: Some log entries lack clear parent-child relationships, leaving certain functions unlinked.
  - ➢ Possible Solution:
    - Leverage the ks-nav database to identify and connect seemingly unlinked functions.
    - Explore automated heuristics to establish missing connections.
- **General Enhancements to ks-nav**
  - ➢ Improve scalability for larger kernels and architectures.
  - ➢ Promote ks-nav from commandline tool and add a web based interface to navigate the code while analyzing graphs. I prefer graphviz layout, but for speed's sake, I'm also considering other javascript based libraries like viz.js

Work in Progress - License: CC-BY-4.0

Thanks
Q&A

ELISA Enabling Linux in Safety Applications

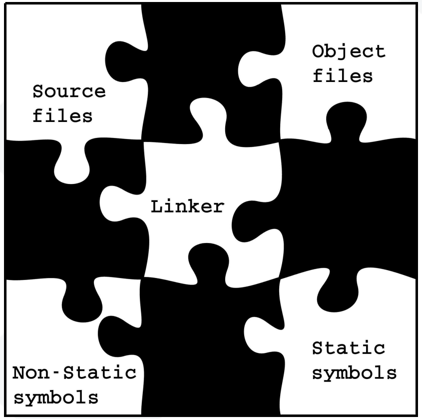
WORKSHOP

Work in Progress - License: CC-BY-4.0

Problems: Duplicate Symbols - Causes

Duplicate Symbol Causes

- **Static Symbols in Separate Compilation Units**:
  - ➢ The linker ignores static symbols, allowing multiple functions with identical names across different object files.
- **Header File Inlines**:
  - ➢ Functions or data defined in headers and compiled in multiple units can result in multiple identical symbols.
  - ➢ Inline directives are suggestions, not guarantees… non-inlined functions become duplicates.
- **Macro-Based Function Variations**:
  - ➢ C files that include other C files (e.g., `compat_binfmt_elf.c`) generate symbols with slight variations due to macros, but retain the same name.
- **Compiler Optimizations**:
  - ➢ Compiler heuristics for inlining and static linkage introduce unpredictable symbol duplication.



ELISA Enabling Linux in Safety Applications

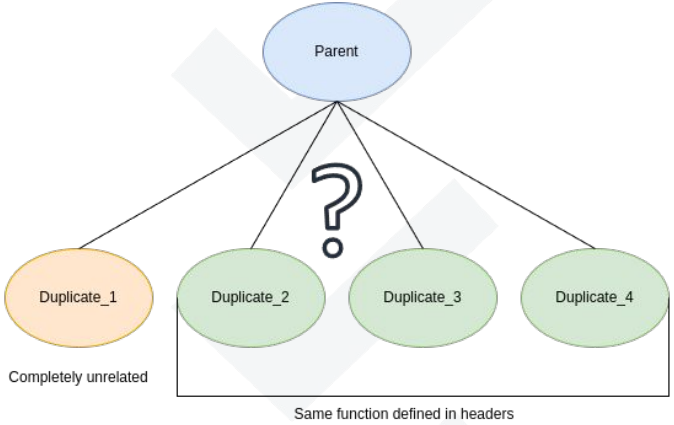
WORKSHOP

Work in Progress - License: CC-BY-4.0

Problem: Duplicate Symbols - Consequences

Impact on ks-nav and Analysis

- **Ambiguous Node Mapping**:
    - ➢ Duplicate symbols can map same function to multiple nodes, causing misinterpretations.
- **Ambiguous names**:
    - ➢ In cases of same-named but distinct functions, can generate confusion when diagrams are read.
- **Testing Challenges**:
    - ➢ Duplicate symbols complicate identifying critical execution paths, skewing coverage and safety verification.



ELISA Enabling Linux in Safety Applications

WORKSHOP

Work in Progress - License: CC-BY-4.0