ELISA
Enabling Linux in Safety Applications

WORKSHOP

LUND 2024

Overall engineering approach to safety for systems involving Linux

Paul Albertella, Codethink
Igor Stoppa, NVIDIA

Summary

- Working group introduction
- Background to current investigation
- Philosophy of overall engineering approach
- Current status
- Next steps
- First principles

Working group introduction

- OSEP: *Open Source Engineering Process* working group
- Developing common processes and frameworks for ELISA
  - Establish a consistent framing / vocabulary for analysis and discussions
  - Develop safety analysis approaches and system models to enable comparison of results
  - Processes for drafting, reviewing and publishing results
- Have historically attempted to focus on safety analysis
  - What kind of *claims* do we want to make about Linux in the context of safety use cases?
  - How can we describe these safety use cases, and analyse the role that Linux plays in them?
- Discussions are frequently more wide-ranging!
  - Processes, methodologies, technical topics, basis for safety claims, competency, etc

Background to current investigation

- Previous ELISA discussions about safety and its application to Linux
  - Safety as a system property
  - Linux development process vs safety standard expectations
  - Varying levels of responsibility for Linux in system designs for Telltale Use Case
- Previous efforts to define approach
  - OSEP proposed approach
  - Safety analysis approach
  - Process for ELISA working group activities
- Contributions from Igor Stoppa
  - Using Linux in a Safe System
  - Checklist for Safety Claims on a Linux System
  - Safety Requirements for a Generic Linux System
  - Interference Scenarios for ARM64 Linux Systems
- System design example from Rail Industry
  - Research Report SIL4 Data Center
  - Suggested as input by Sebastian Hetze
  - Designs for safety-related systems with a 'basic OS' component (not safety-rated)

ELISA Enabling Linux in Safety Applications

WORKSHOP

Philosophy of overall engineering approach

- A system with safety goals might choose to employ the Linux Kernel
  - Safety goals are **specific** to the overall system and to the related application use cases
  - We must expect any solution to be **tailored** to both the **system** and its **goals**
  - Possible that the **only** safety-related role for Linux is as a potential **source of interference**
- How to describe the Linux kernel from a safety perspective?
  - If safety goals and solutions are all **system-specific**, are there any **common** aspects?
    1. Checklist of common **issues** that must be addressed
    2. Analysis of potential **solutions**, and **limitations** or **challenges** that are faced
- Not all issues are necessarily relevant to *every* system or application
  - Safety analysis must clearly state which issues are **relevant**, which are **not**, and **why**
  - Checklist will **expand** and be **refined** over time, so analysis will need **regular review**!

Current status

- Topic: ***Spatial interference via kernel corruption of userspace memory***
  - Based on “KNU bash” example from Igor’s talk at the last workshop: *A Systematic Approach to Using the Linux Kernel in a Safety Scenario*
- Work-in-progress notes in OSEP Wiki
  - https://github.com/elisa-tech/wg-osep/wiki/spatial-kernel-userspace
- Technical inputs being reviewed in a PR
  - https://github.com/elisa-tech/wg-osep/pull/36
  - WIP document: *Linux Memory Management Essentials*
  - WIP document: *Linux Kernel Safety - First Principles*

Next steps

- Continue to review and refine technical input document in PR
  - Identify (and confirm) factual and objectively verifiable statements
  - Use these to inform and refine analysis of issues and potential solutions
- Write up the investigation
  - Currently drafting in Wiki, but will transfer to repository via a PR
  - Goal is to enable and encourage peer review from beyond WG
- Use this as a complementary approach to STPA
- Collaborate with other working groups
  - Starting with input from others on…

First principles - Integrity *(1 of 2)*

1. The vanilla Linux Kernel, by itself, is not sufficient to support safety goals
2. No internal kernel protections prevent interference with (safety-relevant) variable data
3. Any component within the kernel context can generate (cascaded) interference
4. Internal interference can also cascade via components qualified for safety at a unit-level
5. The kernel can interfere with both itself and any part of user-space processes (linear map)
6. No generic solution for very complex systematic corruption of any writable memory
7. Stress testing alone is not sufficient for supporting safety claims
8. Safety claims on the Linux Kernel require both positive testing and focused negative testing

First principles - Integrity *(2 of 2)*

9. Security features that are based on randomisation decrease repeatability of testing
10. Safety claims must be supported by components with the same-or-better safety level
11. Unqualified processes can interfere with qualified ones, through the kernel
12. cgroups/containers/SELinux remove only simple types of interference from user-space
13. In a mixed-criticality scenario, unqualified code represents a safety liability that grows with the frequency of execution (cgroups/containers, LSM/SELinux, mseal, etc.)
14. Hardware features are not a catch-all solution (e.g. ECC Mem ineffective vs unqualified s/w)
15. Multiple sources of interference are too difficult to model reliably: focus on the recipient.
16. Pre-allocation of key resources is not a reliable solution: how to know when it is complete?

First principles - Availability

1. Detecting interference by itself does not necessarily help to control/manage availability
2. Very complex systematic interference makes estimating the probability of a failure unrealistic
3. Stress testing as a means of safety qualification is only realistic for very simple cases (at best)
4. Qualitatively, risk of failure grows with use of unqualified components. For example:
    - invocation of device drivers
    - syscalls
    - memory management
    - Input/Output, e.g. storage, network
    - invocation of Linux Security Module hooks
    - evaluation of cgroups tests
    - presence of non-safety-relevant components triggering the above

Thank you! Questions?

Licensing of Workshop Results

ELISA Enabling Linux in Safety Applications

WORKSHOP