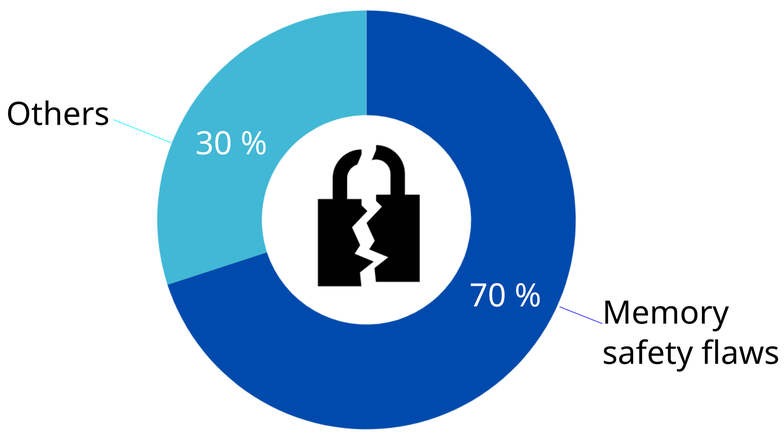
FORCES

FOundations for Reliable, CorrEct and Secure robotic systems

Attilio Discepoli (VUB) Théo Engels (RMA) Robbe de Greef (VUB) Esteban Aguililla Klein (ULB)
Francesco D'Agostino (Thales) Remi Gunsett (Thales) Jonathan Pisane (Thales)
Ken Hasselmann (RMA) Antonio Paolillo (VUB)

THE PROBLEM

Modern robotics still relies on legacy C/C++ code, which allows unsafe memory access. This leads to critical issues like buffer overflows and null pointer dereferences—responsible for high-profile incidents such as Heartbleed and BLASTPASS. In fact, nearly 70% of critical software vulnerabilities stem from memory safety flaws. [1] [2]



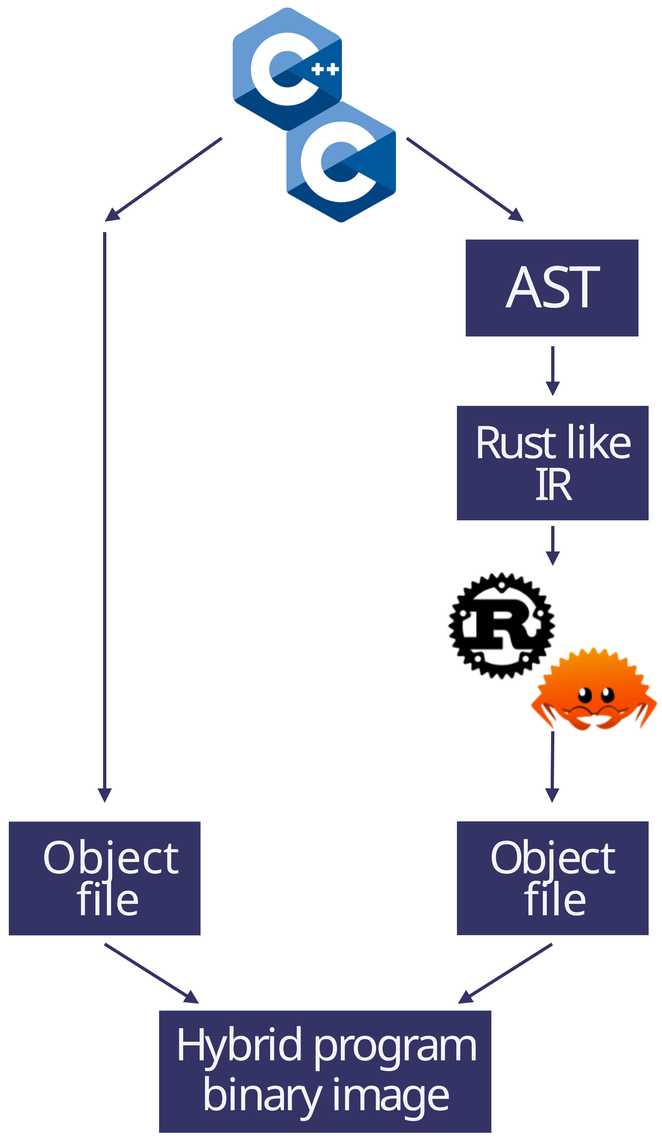
THE SOLUTION

The FORCES project addresses these vulnerabilities by introducing an **incremental transpilation tool** that converts legacy **C/C++ code into memory-safe Rust**. Operating at a fine-grained, function-by-function level, the tool enables a gradual migration—ensuring that critical parts of the code are transformed first while preserving overall system performance.

By leveraging Rust's inherent memory safety, FORCES **helps eliminate common issues** such as buffer overflows and null pointer dereferences. In addition, the project introduces a comprehensive **evaluation framework** that establishes metrics for **correctness**, **security**, **performance** and **maintainability** to assess the effectiveness of the transpilation process. This integrated approach not only fortifies systems against memory vulnerabilities but also paves the way for a smooth modernization of legacy codebases

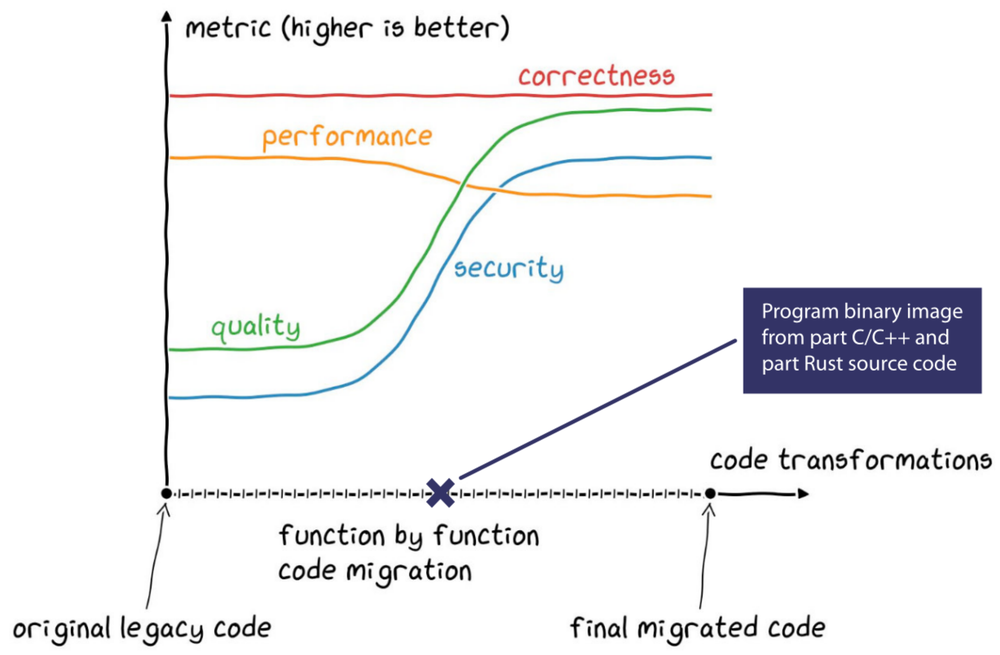
THE METHODOLOGY

Granular Transpiler



The code is **transpiled granularly** - function by function - using the foreign function interface (**FFI**) feature of Rust to gradually integrate the new functions into the resulting codebase so as to gradually translate the entire codebase

Evaluation Framework



Automated testing frameworks and robotic testbeds validate that the transpiled code behaves as intended.

Static and dynamic analysis tools are employed to assess the elimination of memory-related vulnerabilities.

Benchmarking and profiling tools help compare the efficiency of the transpiled code against the original C/ C++ implementation.

Metrics such as code complexity and clarity are tracked to ensure that the refactored code is manageable and easy to extend.

Funded by DEFRA research programme 2024

References

[1] « *We need a safer systems programming language* | MSRC Blog | Microsoft Security Response Center» https://msrc.microsoft.com/blog/2019/07/we-need-a-safer-systems-programming-language/.

[2] «*Secure by Design: Google's Perspective on Memory Safety*» https://research.google/pubs/secure-by-design-googles-perspective-on-memory-safety/.