Formal verification for Rust ported codebases

25th October 2023

Group No: G5

Siddhant Kulkarni	2021A7PS2606G	f20212606@goa.bits-pilani.ac.in	Aditya Bhat	2021A7PS2071G	f20212071@goa.bits-pilani.ac.in
Yash Bhisikar	2021A7PS0483G	f20210483@goa.bits-pilani.ac.in	Nirmal Govindaraj	2021A7PS0441G	f20210441@goa.bits-pilani.ac.in

OVERVIEW

Our objective is to employ formal verification methods, including model checking and symbolic execution, to rigorously assess and make claims about specific memory safety properties of a C/C++ codebase that has been ported to Rust. The analysis involves a detailed examination of ownership and borrowing in Rust and the correctness of "unsafe" code sections. We use specialized tools like Kani Rust Verifier and KLEE Symbolic Execution Engine to ensure the correctness of memory safety properties and functional correctness of certain properties across the C++ and Rust code. Ensuring memory safety will prevent common memory-related errors like buffer overflows, use-after-free, null pointer dereferences and data races.

GOALS

1. Utilize formal verifiers on Rust and C/C++ code.
2. Formulate assertions regarding the properties of the C/C++ and ported Rust code.
3. Comparatively assess memory safety of C/C++ and Rust code.

SPECIFICATIONS

1. **Rust Verification:**
   We will use Kani Rust Verifier, a specialized model checker for Rust. Kani is particularly useful for verifying unsafe code blocks in Rust, which are not checked by the compiler. This will help us verify hybrid C/Rust codebases which may have to use C functions such as malloc() in Rust code for low level applications such as building an OS.
2. **C/C++ Verification:**
   We will employ KLEE, a dynamic symbolic execution engine built on top of the LLVM compiler infrastructure. KLEE will be used to find memory related bugs and vulnerabilities in the C/C++ code, the results of which will be analyzed and compared with the results from Rust verification.
3. **Cross-Language Property Evaluation:**
   Additionally, we will use KLEE to assess the functional correctness of specific properties that span both C/C++ and Rust code. This will help us make claims about the correctness of the ported code from C/C++ to Rust.