Comments of
Open Research Institute, Inc.
#1873 3525 Del Mar Heights Road
San Diego, CA, 92130
United States of America
9 October 2023

Open Research Institute (ORI) is a non-profit research and development organization devoted to open-source digital radio technology. ORI's mission is to provide practical open-source wireless communications solutions for terrestrial and space applications. ORI provides significant workforce development opportunities to the volunteers engaging in open-source technical work.

We understand open-source to mean that everything required to reproduce a software, hardware, or firmware design is freely available to the general public. An open-source design must have free redistribution, allow modifications and derived works, and be non-discriminatory against persons, groups, or fields of endeavor. Open-source work cannot be specific to a particular product, it cannot restrict other software or hardware, and needs to be technology-neutral.

Introduction

ORI agrees that a long-term focus on open-source software security should be a priority to the offices, agencies, and foundations inviting comments in this request. We concur with the assessment that open-source software plays a vital and ubiquitous role not only across the Federal Government, but also in commercial Internet and telecommunications infrastructure.

Central Question

How can the Federal Government drive down the most important systemic risks in open-source software?

By spending money in the near term on software programs that translate important code written in memory unsafe languages into candidate replacement code written in memory safe languages, and then carrying out this work publicly by both selecting and soliciting appropriate source code candidates. Large Language Models with Humans in the Loop (LLM-HIL) would give a possible starting point. Memory safe program translation is an activity very unlikely to be attempted or accomplished by the commercial sector, as there is unclear financial incentive. Memory safe program translation is an activity very unlikely to be attempted or accomplished by the open-source software community, which is fundamentally decentralized. Directly funding translation services for critical software components would drive down systematic risk in an achievable and measurable way.

In the long term, a shift to memory safe languages in the open-source community is already under way. In the marketplace of ideas, and under the pressure of peer-review, languages like Rust have been developed and encouraged and are being adopted.

Rust as an Example

Rust does not provide complete protection against all kinds of bugs or attacks. It is still vulnerable to things like side-channel attacks. As with any language, bugs in third-party libraries can result in security failures. Having said this, Rust is considered to be memory safe, gets consistently high ratings from programmers, and is considered by commercial and non-commercial organizations and communities to be the best option for mission-critical software available today. However, Rust is still pretty far down on the list of most-used computer languages. In order to get the security advantages of Rust, then tools to automate and accelerate refactoring of open-source software components from, for example, C++ to Rust, would be a clearly productive investment.

Codebase Identification

The selection of appropriate codebases at least partially depends on whether or not a memory-safe language translation could be integrated effectively into the product or ecosystem from which the original codebase came. This is where an open nomination process, where an open-source codebase is nominated for translation services funded by the Federal Government, could be very effective. This open nomination process would be in addition to selections of critical software that may come from the offices, agencies, and foundations inviting comments to this RFI. Potential partners for the open nominations process include but are not limited to Open Source Initiative, GitLab, Electronic Frontier Foundation, GitHub, and the Internet Engineering Task Force. It is anticipated that the offices, agencies, and foundations inviting comment in this RFI will nominate many candidates for translation based on their institutional experience and the input they have and will receive.

Testing and Verification

Testing the translated code is a crucial step. It does no good to create a memory-safe program that doesn't work at least as well as the original. Existing code may come with a set of unit tests, verification procedures, and/or functional tests. These tests might be quite comprehensive, or they might be quite cursory, or they might not exist at all. The existing programs that come with a solid test suite are likely to be the programs least in need of translation, so it is very important that the tools developed for translation include tools to facilitate and automate the creation of useful tests.

Automated test development may indeed be a more difficult problem than automatic language translation. They are closely related in that both problems require the inference of the original programmer's intent from the possibly imperfectly-written existing code. It is important that test generation be independent of translation; if the translator makes an error we don't want the same mechanism to result in the generation of tests that treat the error as correct. This means that a full suite of translation tools must solve the underlying problem of intent derivation in two *different* ways.

In evaluating a candidate program for translation, an early step will be to identify and evaluate the existing tests. Since there are no universal standards for automated testing, this step probably cannot be simply automated. Worse, evaluating the power of test procedures is not a solved problem. Various rough metrics, such as code coverage, have been employed with some success, but experience shows these metrics to be inadequate. When tests are developed by humans and evaluated using poor metrics, the perverse incentives created by the flaws in the metrics can result in wasted effort and an inefficient test suite. Automated test development is likely to be subject to the same hazard, only more so. Automatic conversion of an existing test suite into a translated test suite with the power to validate possibly buggy code created by automatic translation may not be possible with presently available technology. A hybrid procedure using humans in the loop may be the best we can do.

Automation, such as automated translation, works on standardized components. Computer languages comply with standards. There are many variations on test implementation, frameworks, practices, and coverage. Delivering "untested" translations of source code is still a big win. Testing can be done manually by the translation team and the status changed to "tested as least as well as the original", or the translated code can be clearly marked as "untested" and then shared as quickly as possible. Manual or customized testing means understanding the original source code testing process and then duplicating that process with the translated codebase. The difficulty of this job may be reduced by a growing number of tools like LLMs.

Respectfully,

Paul Williamson
ORI Factotum

Michelle Thompson
CEO ORI