Advantages and pitfalls of migrating from C/C++ to Rust with respect to memory safety bugs

Bruno Martins

bm2787

Section 1: Abstract

Systems software has long been dominated by the use of the low level programming languages C and C++: the Linux kernel, game engines and every major web browser are some of the most prominent examples of such systems. However, while C and C++ are the go-to languages to extract the most performance out of the hardware, they lack important safety features to prevent potentially catastrophic errors such as null pointer dereferences and buffer overruns. While such bugs in C/C++ code can potentially be detected by modern static analysis tools, Rust, a new programming language by Mozilla, claims to have eliminated entire classes of bugs at the language level by leveraging its unique type system. This paper looks into these claims, first by performing a review on a state-of-the-art static analyzer, Infer, then by evaluating the use of Rust in a real world stable production system, EPICS. Known race conditions, null dereferences and buffer overrun bugs in EPICS were reimplemented in Rust to evaluate both Rust's ability to detect them and how easy such reimplementations were. The reader is expected to learn about Rust, its strengths in bug prevention and the feasibility and potential pitfalls of migrating an existing C/C++ program into Rust.

Section 2: Introduction

(This section will expand on the problem statement given in the abstract. It will explain the different classes of bugs being considered in this paper and why Rust might be a good alternative to C/C++ for systems programming. This section will also explain how the paper is structured. Also, the two main questions investigated by this paper will be posed:

1. Does Rust prevent bugs?

2. Is it straightforward to translate C/C++ programs into Rust?)

Section 3: Background

Subsection 3.1: Formal reasoning(state of the art)

(Here I will present a review of hoare logic [1] and separation logic [2], which are used in the Infer [3] static analyzer. Then I will talk about Infer itself. The point is to contrast the approach that has to be taken to analyze a C/C++ program (i.e. running a whole separate program that may/may not catch bugs) with the safety provided at the language level by Rust.

Rust is not completely free from all bugs, of course, so I will mention efforts to verify Rust "unsafe code" by exhaustive test generation [4], regular program verification by symbolic execution [5], type checker verification by fuzzing [6], and formal verification by using separation logic [7]. I think this paragraph could be an entire section maybe.)

Subsection 3.2: Rust

Rust is a new systems programming language, developed by Mozilla, focused on safety, speed and concurrency [8]. The first stable release of the language, dubbed version 1.0, was released on May 2015. Rust development was driven chiefly by the concurrent development of the Servo web browser engine, also by Mozilla. Rust's advanced type system helped Servo developers to better address common security issues while still producing code with good performance and similar memory usage [9].

Rust's strength lies in its type system.

(In this section I will expand on Rust's type system, provide a few code snippets so the reader can get a feel for the language, and then show how Rust catches (or prevents) null deref, buffer overrun, resource leaks)

Subsection 3.3: EPICS - Experimental Physics and Industrial Control System

The EPICS framework [10] is a set of software libraries and tools designed to fulfill the role of a SCADA (supervisory control and data acquisition) system for big science facilities and industrial plants. EPICS has a distributed nature: its main component is the IOC (Input/Output Controller), which is responsible for communicating with physical devices and making their accepted commands and captured data available to the network via PVs (Process Variables). The PVs in an IOC are considered part of a Distributed Run Time Database. Other EPICS components, like graphical user interfaces, data archivers and user scripts, interact with the IOCs (and therefore the devices they control) by reading from and writing to the available PVs on the network, via one of two existing protocols: Channel Access [11] or PV Access [12]. Channel Access, the older of the two, is capable of transporting scalar and array values only, along with their metadata. PV Access, on the other hand, can transport arbitrarily structured data. A simplified diagram of EPICS' architecture is shown in Figure 1.

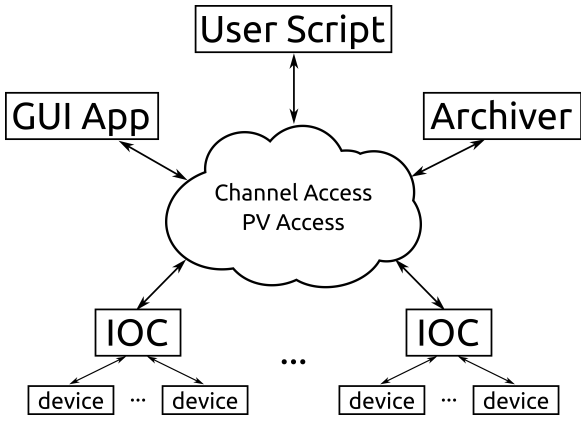

Figure 1: Simplified EPICS architecture

EPICS is used in different kinds of scientific facilities across the world [13]: particle accelerators, observatories, telescope arrays, research fusion reactors and many others. Given its widespread use and the critical rote it performs, it is of paramount importance that its central component, the IOC, doesn't crash.

(I can expand more if needed. I feel that this section is just for the reader to get a feel for the complexity of the software being modified to use Rust)

Secion 4: Methodology

(Here I will explain that I developed a small web app to quickly classify EPICS bugs going back to 2009. I will post screenshots of my tools, explain the criteria that I used to select bugs, show how many bugs I assigned to each category, explain which bugs and why I chose them to be reimplemented in rust.)

Section 5: Results

(Here I will present the selected bugs, potentially their simplified version in C/C++, the translated version in Rust and Rust's compiler output. I will evaluate if the known bug was caught by the compiler and if the message given is clear. I will also present my evaluation on whether the translations were sufficiently straightforward to be done.)

Section 6: Conclusion

(In the conclusion section, my expected conclusion is that Rust will have caught all given bugs, but that the translation won't be as straightforward as it could be, given the reports from other people about Rust's steep learning curve)

References

[1] C. A. R. Hoare, “An Axiomatic Basis for Computer Programming,” *Commun ACM*, vol. 12, no. 10, pp. 576–580, Oct. 1969.

[2] J. C. Reynolds, “Separation Logic: A Logic for Shared Mutable Data Structures,” in *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, Washington, DC, USA, 2002, pp. 55–74.

[3] R. van Tonder and C. L. Goues, “Static Automated Program Repair for Heap Properties,” in *Proceedings of the 40th International Conference on Software Engineering*, New York, NY, USA, 2018, pp. 151–162.

[4] J. Toman, S. Pernsteiner, and E. Torlak, “Crust: A Bounded Verifier for Rust (N),” in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 75–80.

[5] M. Lindner, J. Aparicius, and P. Lindgren, “No Panic! Verification of Rust Programs by Symbolic Execution,” in *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, 2018, pp. 108–114.

[6] K. Dewey, J. Roesch, and B. Hardekopf, “Fuzzing the Rust Typechecker Using CLP (T),” in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 482–493.

[7] R. Jung, J.-H. Jourdan, R. Krebbers, and D. Dreyer, “RustBelt: Securing the Foundations of the Rust Programming Language,” *Proc ACM Program Lang*, vol. 2, no. POPL, pp. 66:1–66:34, Dec. 2017.

[8] N. D. Matsakis and F. S. Klock II, “The Rust Language,” in *Proceedings of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology*, New York, NY, USA, 2014, pp. 103–104.

[9] B. Anderson *et al.*, “Engineering the Servo Web Browser Engine Using Rust,” in *Proceedings of the 38th International Conference on Software Engineering Companion*, New York, NY, USA, 2016, pp. 81–89.

[10] L. R. Dalesio, A. J. (Los A. N. L. Kozubal, and M. R. (Argonne N. L. Kraimer, “EPICS architecture,” Los Alamos National Lab., NM (United States), LA-UR-91-3543; CONF-911116-9, Jan. 1991.

[11] “EPICS Channel Access 4.13.1 Reference Manual.” [Online]. Available: https://epics.anl.gov/base/R7-0/2-docs/CAref.html. [Accessed: 24-Feb-2019].

[12] “pvAccess Protocol Specification.” [Online]. Available: http://epics-pvdata.sourceforge.net/pvAccess_Protocol_Specification.html. [Accessed: 24-Feb-2019].

[13] “Projects - EPICS Controls.” [Online]. Available: https://epics-controls.org/epics-users/projects/. [Accessed: 24-Feb-2019].