1-1: Rust Introduction

Artem Pavlov, TII, Abu Dhabi, 03.05.2024

About me

- Artem Pavlov [Артём Павлов]
- Started programming in 2006. Languages:
  Pascal, Delphi, C, Python, JavaScript, Go, Rust
- Programming in Rust since 2016
- Education:
  - Moscow Institute of Physics and Technology (BSc, MSc)
  - Skolkovo Institute of Science and Technology (PhD)
- E-mail: newpavlov@gmail.com Telegram: @newpavlov

My open source work

- GitHub: https://github.com/newpavlov

- Project Lead of the RustCrypto project

- Member of the rust-random team

- Various contribution to the Rust ecosystem

My professional work

- Team Lead, Department of System Software Development, Quantom LLC, Moscow, Russia

- https://quantom.info

- Development of RDBMS and cryptographic products using Rust

- One of projects (QSS) is certified as a cryptographic product by the Federal Security Service (FSB)

About audience

- General programming experience
- Experience in systems programming and "low-level" languages
- Do you know Rust?

About this workshop

- Dates: from June 3 until June 12
- Day schedule:
  - • 9:00-10:30 Slot 1 (Theory)
  - • 10:30-10:45 Break
  - • 10:45-12:30 Slot 2 (Practice)
  - • 12:30-13:30 Dinner
  - • 13:30-15:00 Slot 3 (Theory)
  - • 15:00-15:15 Break
  - • 15:15-17:00 Slot 4 (Practice)

- Rust Introduction
  (14-18 April)
- Cryptography in Rust
  (21-22 April)
- Project Day
  (23 April)
- Project Presentation
  (23 April)

Recommendations

- For this workshop it's recommended to use Linux OS on x86-64 CPU

- Recommended IDE is Visual Studio Code

- Windows and Mac systems, as well as other IDEs are acceptable, but I may not be able to help with all issues

The Rust programming language

- Fast, Reliable, Productive. Pick Three.
- A memory-safe* systems programming language
- Zero-cost abstractions (pay only for what you use)
- Modern tooling and library management
- The most loved language for 8 years in a row
- The only language outside of C allowed in the Linux kernel

* In the safe language subset, assuming unsafe code in used libraries and the Rust compiler are correct

A brief history of Rust

- Started in 2006 by Mozilla Research employee Graydon Hoare

- Drastic change in direction in 2012

- 1.0 release on May 15, 2015

- Formation of the Rust Foundation on February 8, 2021

Systems programming language

- Compiles to binary code

- Zero-cost abstractions

- Minimal runtime

- No garbage collector and hidden allocations

- Static dispatch (unless explicitly opted out)

- Anywhere you use C, you can use Rust!*

Safety

- Rust enforces memory and data race safety in the safe subset of the language

- Concepts of ownership and borrowing play a big role in it

- Forget about:

  - Double free

  - Use after free

  - Iterator invalidation

  - Null pointer dereferencing

Misconceptions about unsafe

- Rust provides an "escape hatch" in the form of unsafe blocks

- Using unsafe in a library does not mean that its users are not memory safe!

- Correctly written libraries encapsulate unsafe code in safe APIs, which are impossible to misuse

Expressive and rigorous type system

- Rust type system helps to make invalid states unrepresentable

- Modern features from functional languages: sum types, exhaustive pattern matching, etc.

- "Strongly typed" generics system based on traits (interfaces on steroids)

Safety against data races

- Rust prevents data races at compile time

- But not race conditions in general!

- Done by the type system together with special traits: Send and Sync

- This enables the "fearless concurrency", i.e. it allows to efficiently write and refactor multi-threaded programs

Modern tooling

- Compiler toolchain management system: rustup
- One "canonical" build system: cargo
- Central repository for libraries (crates): crates.io
- Central repository for documentations: docs.rs
- Standard library docs: https://doc.rust-lang.org/stable/std
- Playground: https://play.rust-lang.org
- Language Server Protocol implementation: rust-analyzer

Standard Cargo tooling

- Cargo includes standard tools:
  - Testing: cargo test
  - Benchmarking: cargo bench
  - Formatting: cargo fmt
  - Docs generation: cargo doc
  - Advanced linter: cargo clippy

Cargo extensions

- Cargo can be extended by installing plugins
- Installation is done using cargo install cargo-plugin
- Plugin examples:
  - cargo tree: visualize dependency graph in a tree-like format
  - cargo audit: audit dependencies for crates with security vulnerabilities
  - cargo license: collect license of dependencies
  - cargo deb: automatically generate Debian packages

Rust is a *practical* language

- Rust is NOT a language with built-in verification of programs

- Rust does not yet have a full formal language specification and specification of memory model in particular

- On robustness spectrum, it's above C/C++ and many GC languages, but below Coq, Idris, and SPARK

Rust is not an ideal language

- Rust and LLVM have bugs

- Rust supports only targets supported by LLVM

- Some important features are either missing, or available only on Nightly (e.g. const generics, specialziation)

- Rust has some questionable features (e.g. async)

Why use Rust for cryptography?

- 70% of CVEs are caused by memory issues as reported by Microsoft and Google

- Implementation of cryptographic algorithms is a relatively small part of implementing a cryptographic software

- Bugs like Heartbleed are much less likely in Rust

Questions?