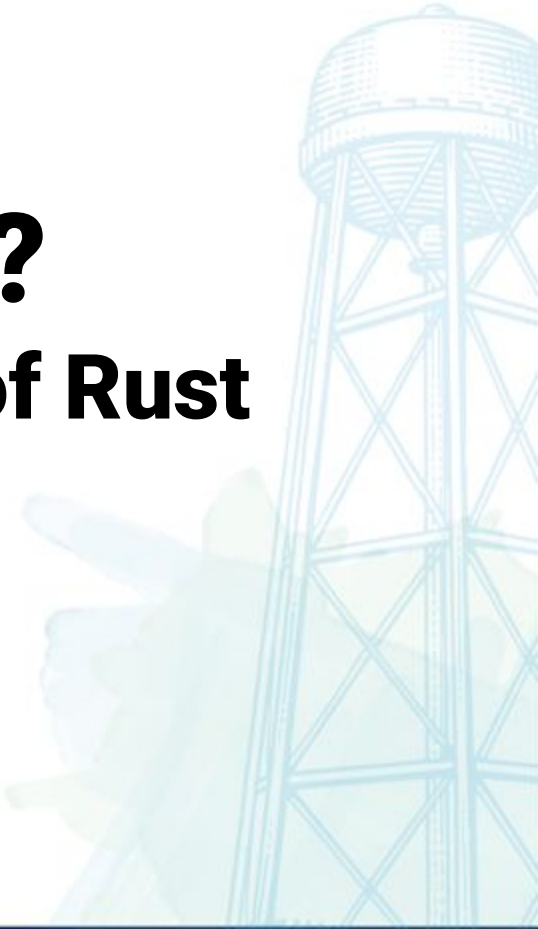
In Rust We Trust?
A Closer Look to the Safety of Rust Crates

Final Project of ECS 289C

Spring 2024

Muhammad Hassnain
Parnian Kamran



UCDAVIS

Outline

- Motivation
- Crates & Tool Selection
- Comparison of Tools
- Limitations
- Some Problems and Solutions
- Runtime and Memory Consumption Analysis
-  Discussion

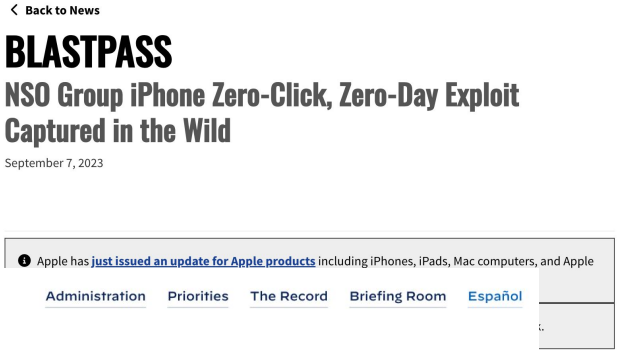
Motivation

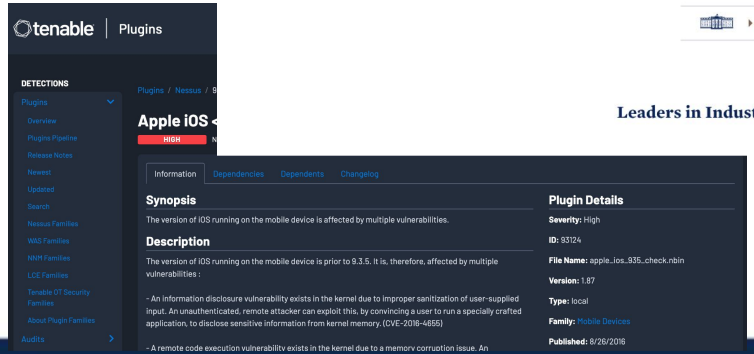
- Rust?

Not this ❌

This ✅

Motivation





THE WHITE HOUSE

FEBRUARY 26, 2024

Press Release: Future Software Should Be Memory Safe

ONCD › BRIEFING ROOM › PRESS RELEASE

Leaders in Industry Support White House Call to Address Root Cause of Many of the Worst Cyber Attacks

views, and insight from the ESET security comm

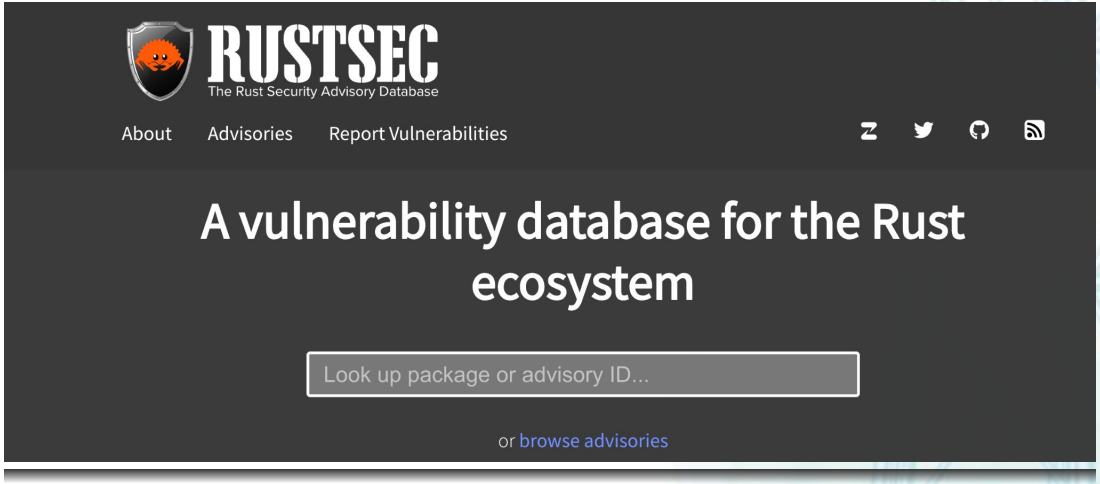
WeLiveScience FEATURED TOPICS

SQL Slammer had brought the
otable worm.

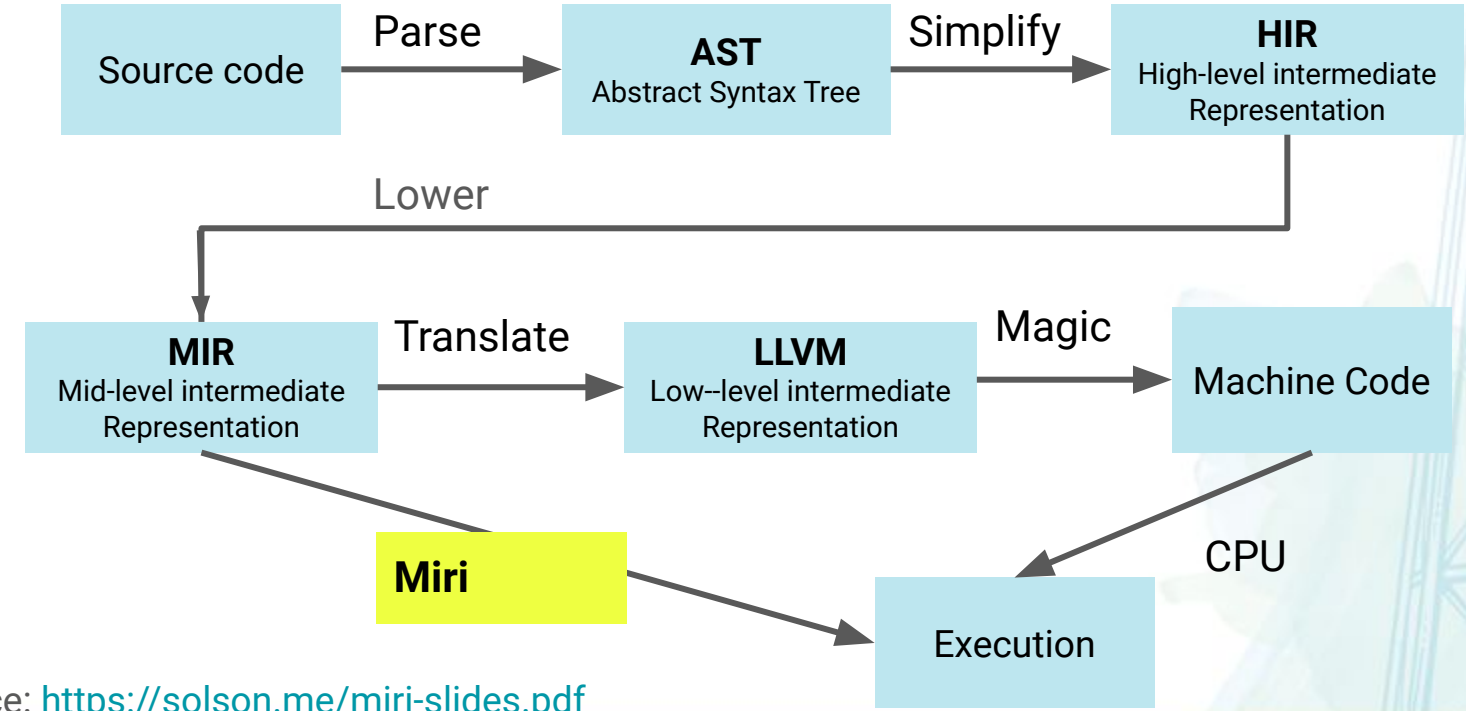
On Saturday 25th January 2003, the internet was hit by a rapacious computer worm now known as SQL Slammer. Spreading like wildfire over the internet via a bug in a version of Microsoft SQL, it is believed to have infected over 75,000 machines within a matter of minutes. Globally, over 250,000 computers were thought to have been affected.

Crates Selection

- Rust Sec Database
  - Critical (20 crates)
  - High (10 crates)
- Safe (10 crates)
  - Identify false positives

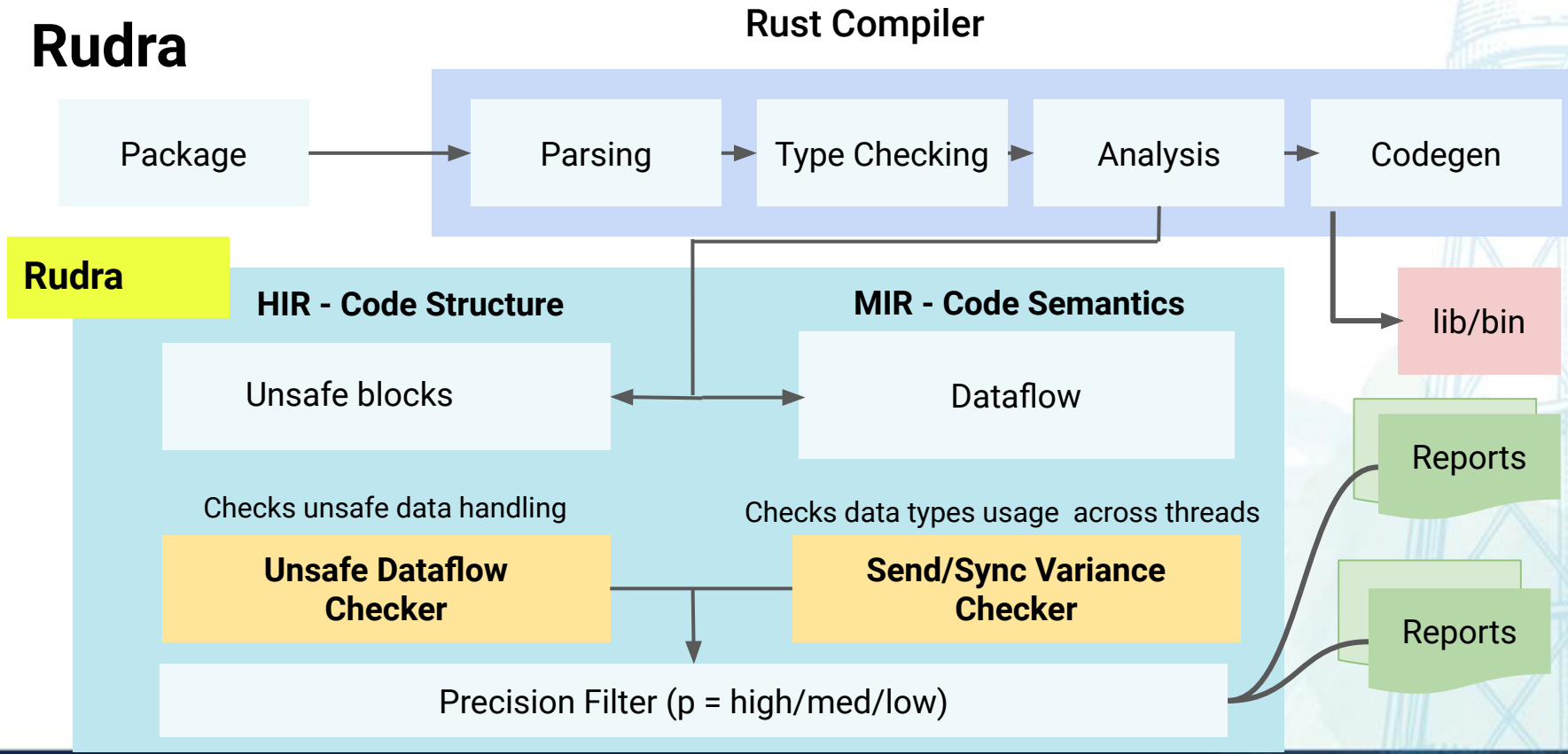


Miri: Interpreter for Rust Mid-level Representation

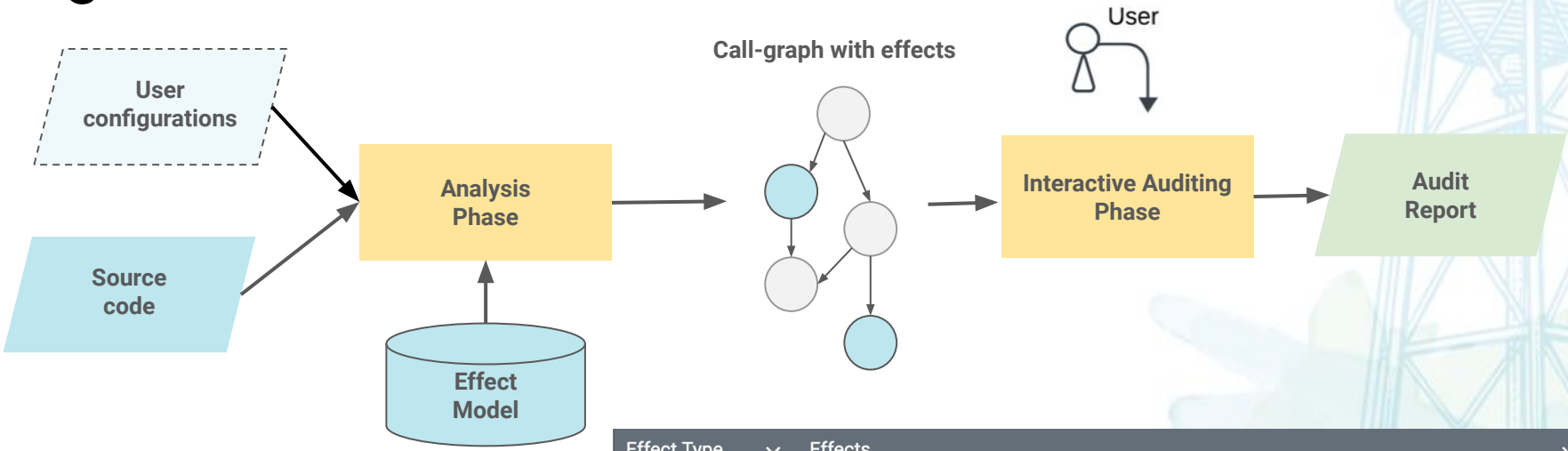


Reference: https://solson.me/miri-slides.pdf

Rudra


Rust Compiler
Package
Parsing
Type Checking
Analysis
Codegen
Rudra
HIR - Code Structure
MIR - Code Semantics
Unsafe blocks
Dataflow
lib/bin
Reports
Checks unsafe data handling
Checks data types usage across threads
Unsafe Dataflow Checker
Send/Sync Variance Checker
Reports
Precision Filter (p = high/med/low)

Cargo-Scan



Effect Type	Effects
Unsafe Effects	FFICall, FFIDecl, StaticExt, StaticMut, UnsafeCall, UnionField, RawPointer
System Effects	std::fs, std::io, std::os, std::ffi, std::net, std::env, std::arch, std::path, std::mem, std::simd,
(SinkCall)	std::panic, std::process, std::backtrace, std::intrinsics, libc, winapi
High-order Effects	FnPtrCreation, ClosureCreation

Big Table

- List of crates
- Tools results analysis
- First sheet -> Rust analyzers

Let’s take a poll

Limitations

Tool	Limitation	Detail
Miri	API Support	No support for hardware APIs, filesystems, FFI, network
	Warnings	Shows some vulnerabilities as warnings
Rudra	Compiler Version	Uses rustc 1.58.0, not supported by many crates
	Parsing Errors	Fails to parse Cargo.toml for most crates (does not support “^”)
Cargo-Scan	False Positive	Outputs all side effects, so lot's of false positives
	Macros	Does not support Macros.

Data appropriation

Rudra and Rust Version Mismatch

```
2024-06-03 06:38:28.467686 |INFO | [rudra-progress] Running cargo rudra
2024-06-03 06:38:34.266699 |INFO | [rudra-progress] Running rudra for target lib:lz4-sys
error: package `cc v1.0.98` cannot be built because it requires rustc 1.63 or newer, while the currently active rustc version is 1.58.0-nightly
2024-06-03 06:38:43.235907 |ERROR| [rudra-progress] Finished with non-zero exit code
```

Solution?

Data appropriation

Rudra and Rust Version Mismatch

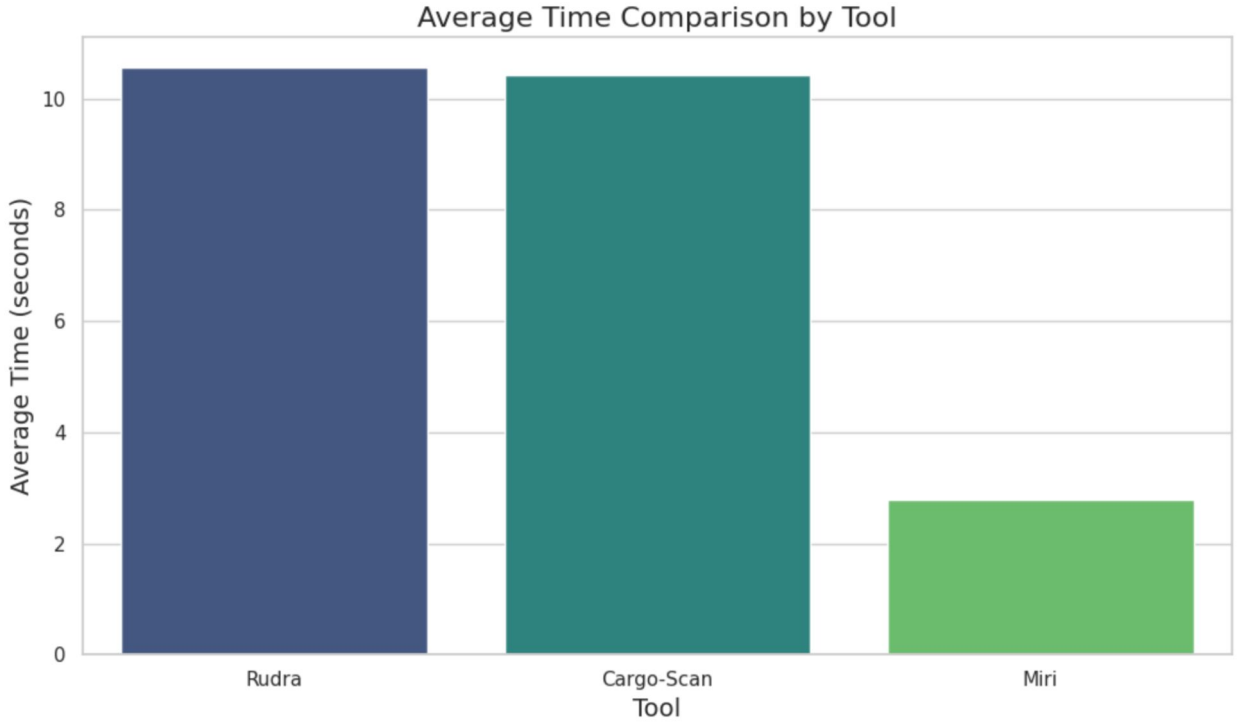
```
2024-06-03 06:38:28.467686 |INFO | [rudra-progress] Running cargo rudra
2024-06-03 06:38:34.266699 |INFO | [rudra-progress] Running rudra for target lib:lz4-sys
error: package `cc v1.0.98` cannot be built because it requires rustc 1.63 or newer, while the currently active rustc version is 1.58.0-nightly
2024-06-03 06:38:43.235907 |ERROR| [rudra-progress] Finished with non-zero exit code
```

Solution

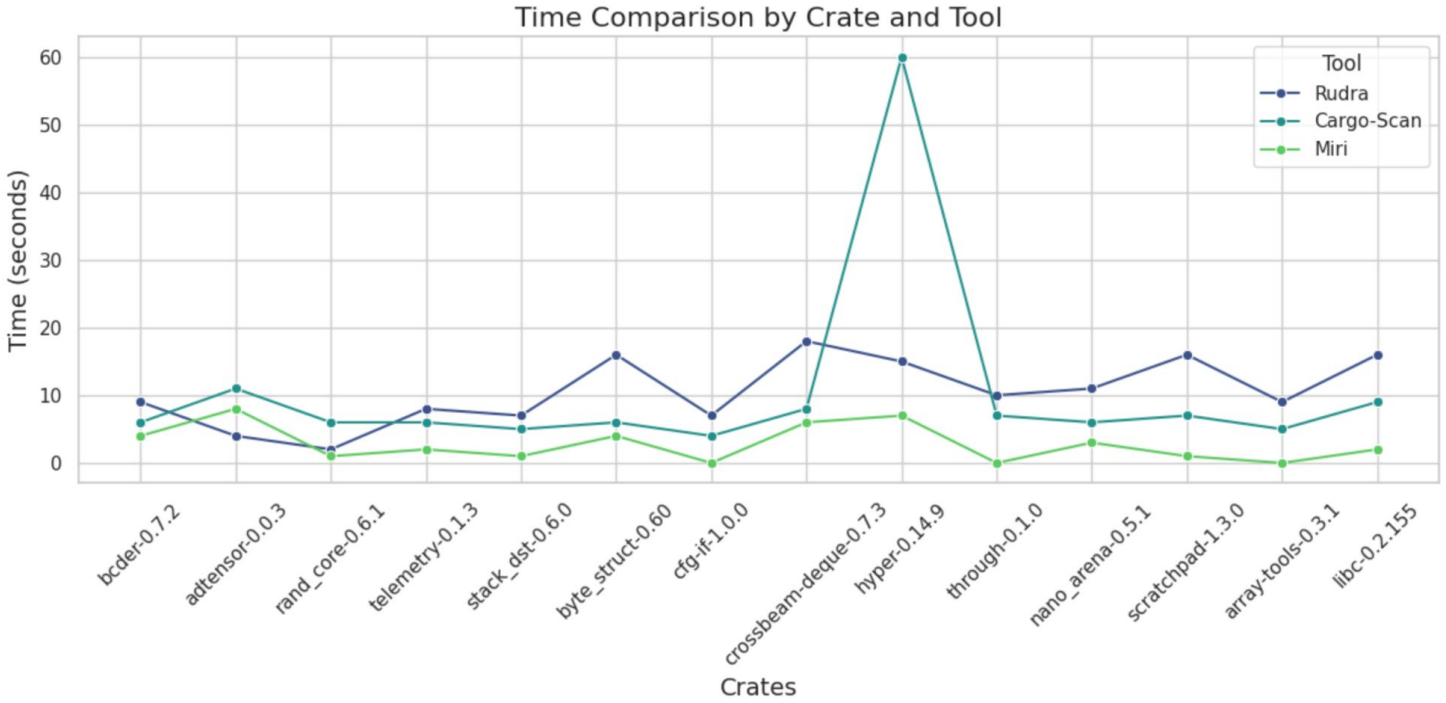
```
2024-06-03 06:39:54.568213 |INFO | [rudra-progress] Running cargo rudra
2024-06-03 06:39:59.702995 |INFO | [rudra-progress] Running rudra for target lib:lz4-sys
warning: use of deprecated type alias `gcc::Config`: gcc::Config has been renamed to gcc::Build
 --> build.rs:6:29
  |
6 |     let mut compiler = gcc::Config::new();
  |                            ^^^^^^
  |
  = note: `#[warn(deprecated)]` on by default

warning: use of deprecated associated function `gcc::Build::new`: crate has been renamed to `cc`, the `gcc` name is not maintained
 --> build.rs:6:37
  |
6 |     let mut compiler = gcc::Config::new();
  |                                     ^^^
```

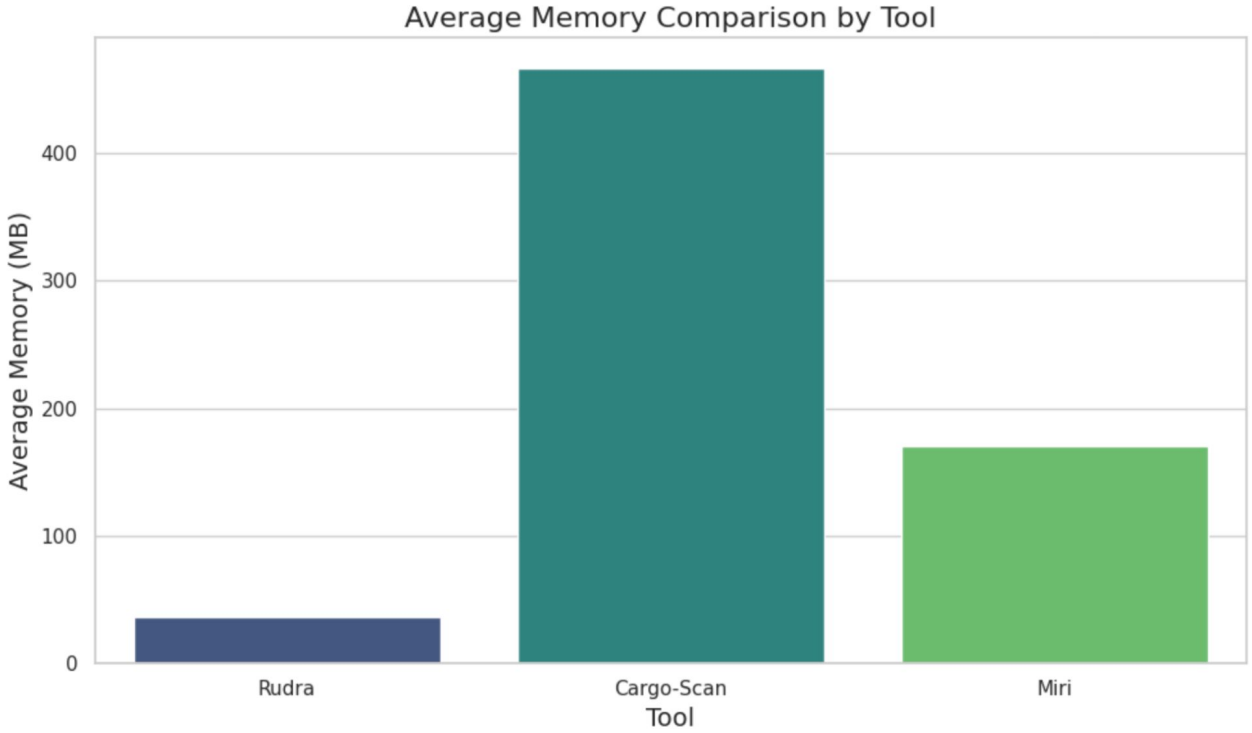
Execution Time



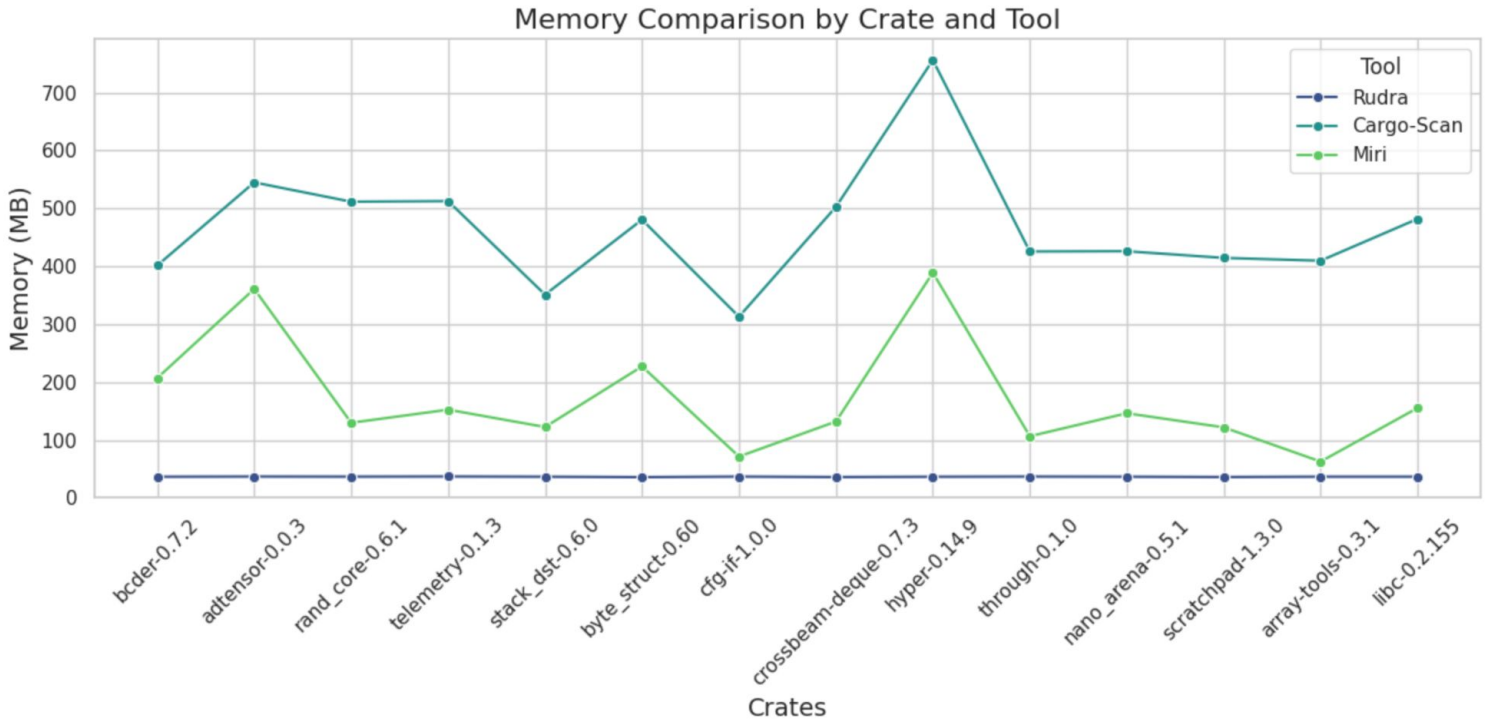
Execution Time - Deeper Dive



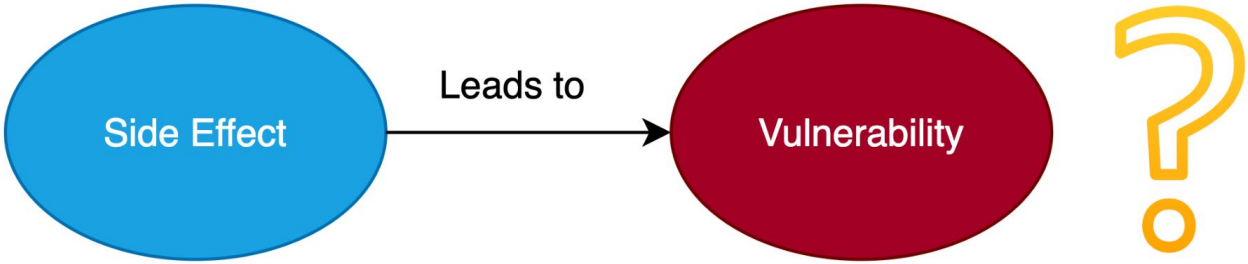
Memory Consumption



Memory Consumption - Deeper Dive



Discussion



Thank You

Do not trust any code you download from internet (including LLMs)

Poll Result

Outline

5 Level of the Project

Motivation

Crates & Tool Selection

Comparison of Tools

Limitations

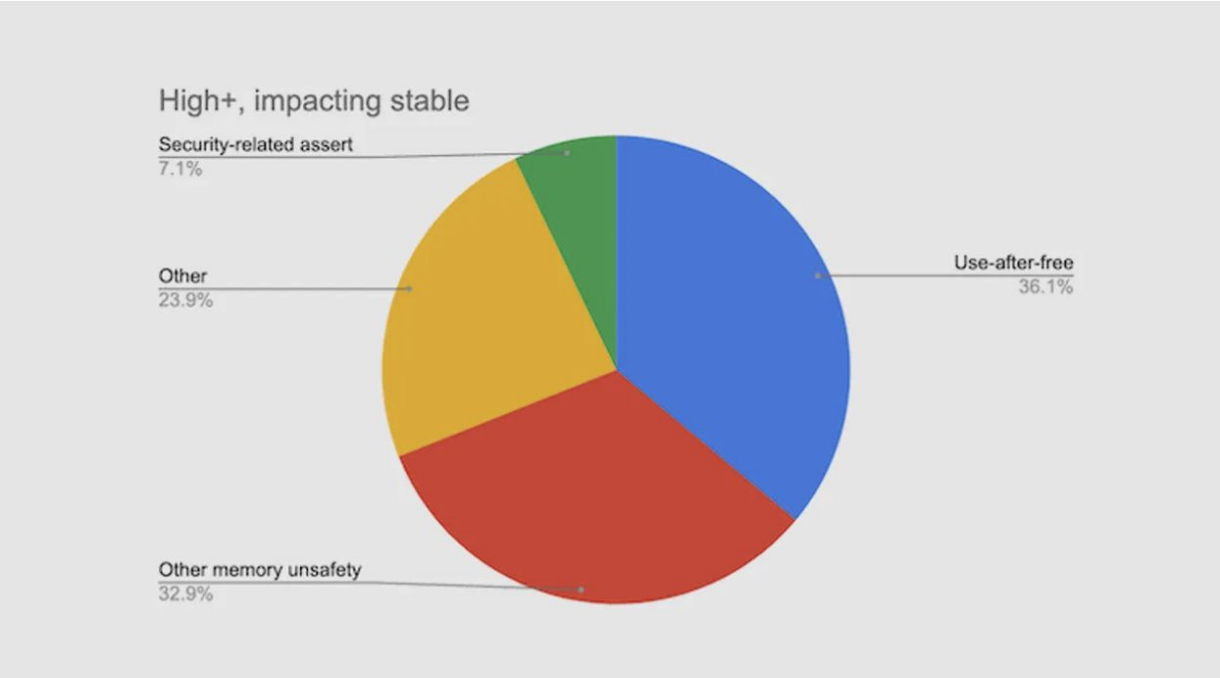
Some Problems and Solutions

Runtime and Memory Consumption Analysis

Discussion

Motivation

- Rust
- Why Rust?

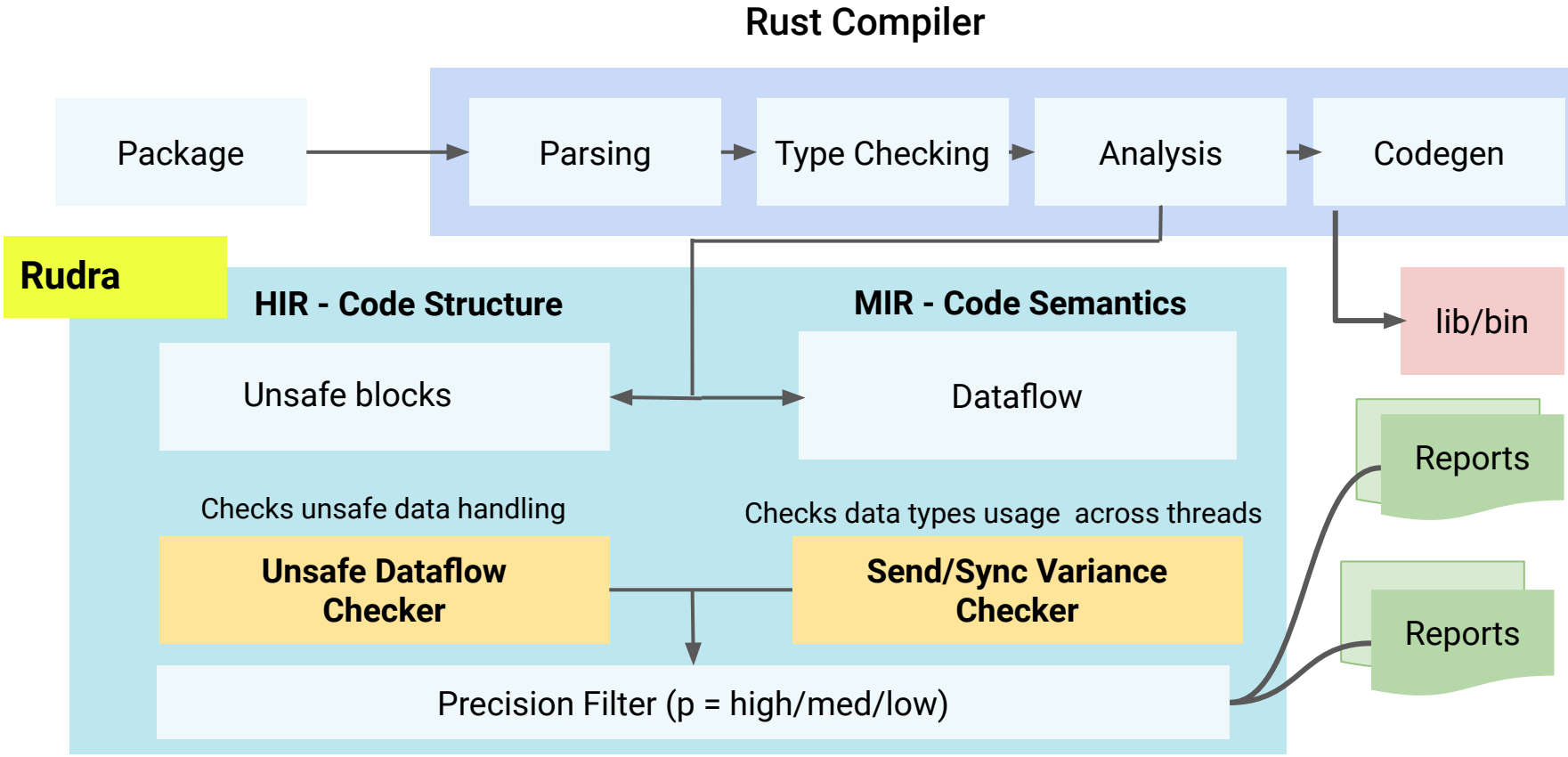


5-Levels of Research Project

- Research area: PL
- Research subarea: Software security, Software reliability
- Research topic: Security in Rust Ecosystem
- Research problem: Are Rust libraries a threat for Rust codes security and robustness
- Research solution: Evaluate the proficiency of Rust analyzers

Motivation

- Rust
- Code written in Rust guarantees:
  - Runtime performance of traditional system languages like C/C++
  - Memory Safety
    - No Dangling Pointers
    - No Garbage Collector
    - No Use after free or out of bounds
  - Type Safety
- Rust allows certain operations using “unsafe” keyword
- Rust compiler cannot guarantee safety in “unsafe” blocks


Rust Compiler
Package
Parsing
Type Checking
Analysis
Codegen
Rudra
HIR - Code Structure
MIR - Code Semantics
lib/bin
Unsafe blocks
Dataflow
Reports
Checks unsafe data handling
Checks data types usage across threads
Unsafe Dataflow Checker
Send/Sync Variance Checker
Reports
Precision Filter (p = high/med/low)