

Reversing Rust Binaries



*One Step Beyond **strings***

Cindy Xiao

(she/her, they/them)

Senior Security Researcher, CrowdStrike

June 28, 2024

Introduction



- Are you a seasoned reverse engineer, but you tremble when a Rust binary lands on your desk?
- When you encounter a Rust binary, do you just run `strings` on it and hope for the best?

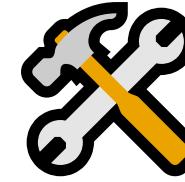
One step beyond **strings**

Let's use **strings** as an approachable starting point for exploring reversing Rust binaries.

Today, we will do the following:

- Build a working Rust downloader.
- Learn about string metadata inside our downloader.
- Find the entry point of our downloader, using strings.
- Know where to find more information about the Rust runtime, standard library, and third-party libraries.

Setup and tools

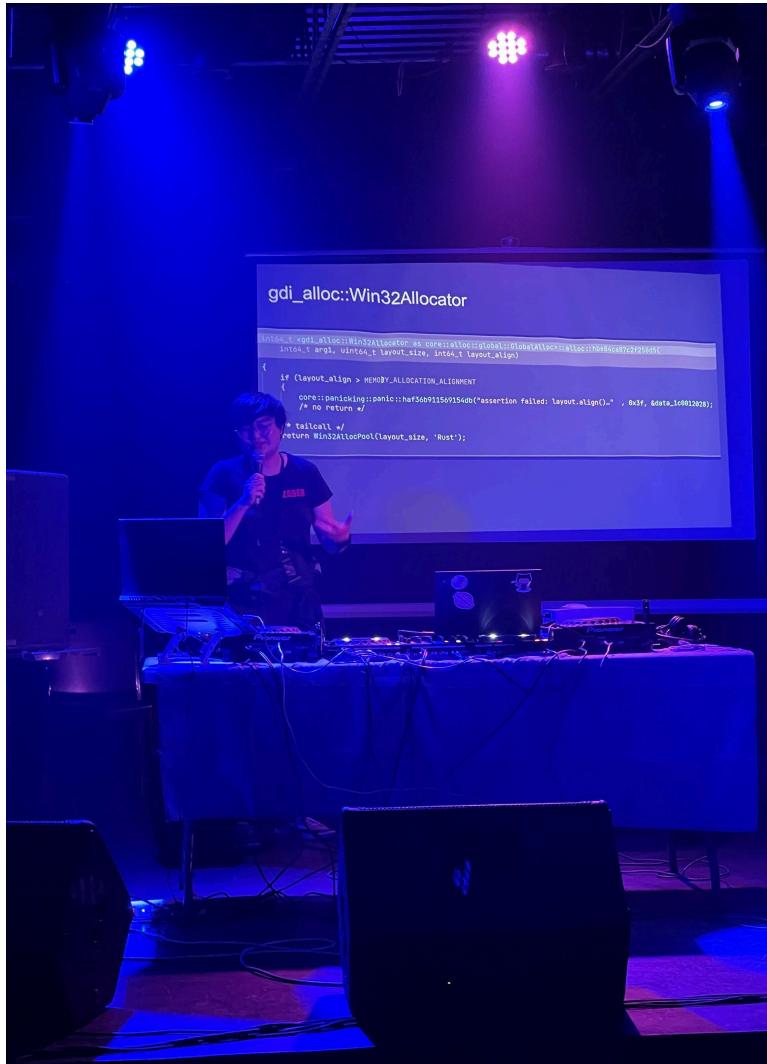


- 🔗 Full setup instructions + prebuilt VM:

→ github.com/cxiao/rust-reversing-workshop-recon-2024 ←

- Rust toolchain
 - Go to <https://rustup.rs/> and follow the instructions for your OS!
 - Make sure you can run `rustup` and `cargo` from your terminal, after installation
- Disassembler / decompiler
- Internet connection

My background



REcon 2023 lightning talk about Rust in the Windows kernel (thanks Sergei for the pic)

- I currently do malware analysis, reverse engineering, and cyber threat intelligence
- I used to be a C/C++ developer, and I'm interested in approaching things from the developer's perspective
- We were seeing more Rust malware in our analysis queue, and we needed practical skills to deal with them

Rust Malware-O-Meter



A selection of public reporting about new Rust malware in 2023 and 2024:

- 2023-04-21 - Jamf - BlueNoroff APT group targets macOS with 'RustBucket' Malware
- 2023-04-04 - Trustwave - Rilide: A New Malicious Browser Extension for Stealing Cryptocurrencies
- 2023-05-19 - Trend Micro - Rust-Based Info Stealers Abuse GitHub Codespaces
- 2023-07-19 - Palo Alto Networks - P2PIInfect: The Rusty Peer-to-Peer Self-Replicating Worm
- 2023-11-09 - Bitdefender - Hive Ransomware's Offspring: Hunters International Takes the Stage
- 2023-12-21 - Seqrite - Operation RusticWeb targets Indian Govt: From Rust-based malware to Web-service exfiltration
- 2024-01-18 - Google - Russian threat group COLDIVER expands its targeting of Western officials to include the use of malware
- 2024-01-29 - Synacktiv - KrustyLoader - Rust Malware Linked to Ivanti ConnectSecure Compromises
- 2024-02-08 - Bitdefender - New macOS Backdoor Written in Rust Shows Possible Link with Windows Ransomware Group



Collected over at github.com/cxiao/rust-malware-gallery

Rust Malware-O-Meter



CrowdStrike Falcon Intelligence reporting:

- 13 intelligence reports on Rust malware in 2022.
 - Mostly just used by a few ransomware developers (ALPHA SPIDER, HIVE SPIDER)
- 29 intelligence reports on Rust malware in 2023.
 - 00 2
- 13 intelligence reports on Rust malware so far in 2024.
 - Downloaders, loaders, backdoors, ransomware, info stealers, etc.

It's becoming a problem

- Frequency - see previous slides
- Scope - used for both malware delivered in targeted attacks, and commodity malware deployed opportunistically
- Sophistication - ranging from full-featured ransomware to very simple downloaders

We're at the point where "written in Rust" is not really a distinguishing feature anymore.

- (i.e. it's time to stop naming Rust malware with names like *Rust(DESCRIPTION_OF_FUNCTIONALITY)*. Please, no more *RustyStealers*.)

How much Rust do I need to know?



We can't learn all of Rust today, but we will need to understand some Rust source code.

- Rust is a very different language from C.
 - We need to take the source emitted by our decompiler and go one abstraction level up.
- The Rust standard library is written in Rust, and so is the Rust compiler.
 - Sometimes, we will need to look at those when trying to figure out how something works.

Setup and tools



- Rust toolchain
 - Go to <https://rustup.rs/> and follow the instructions for your OS!
 - Make sure you can run `rustup` and `cargo` from your terminal, after installation
 - Disassembler / decompiler
 - Internet connection
-  Full setup instructions + prebuilt VM: github.com/cxiao/rust-reversing-workshop-recon-2024

Terminology and tools: *rustup*



- **rustup**: The Rust toolchain installer and toolchain manager.
 - Update installed toolchains
 - ⇒ e.g. “I want to update from Rust 1.77.0 to Rust 1.77.1”
 - Manage cross compilation support
 - ⇒ e.g. “I want to be able to cross compile to Windows”
 - Install multiple Rust toolchain versions side by side
 - ⇒ e.g. “I want to install the latest nightly build of Rust, so I can try out this cool new language feature”

Terminology and tools: *rustup*



rustup update makes it trivial for developers to grab the newest stable version of Rust, which comes out every 6 weeks! A small icon of a computer monitor displaying a graphical user interface with blue and white elements.

```
$ rustup update
info: syncing channel updates for 'stable'
info: downloading component 'rustc'
info: downloading component 'rust-std'
info: downloading component 'rust-docs'
info: downloading component 'cargo'
info: installing component 'rustc'
info: installing component 'rust-std'
info: installing component 'rust-docs'
info: installing component 'cargo'
info: checking for self-update
info: downloading self-update

stable-x86_64-unknown-linux-gnu updated - rustc 1.77.2 (25ef9e3d8 2024-04-09)
```

Building our own Rust binary

Building our very own downloader ❤️

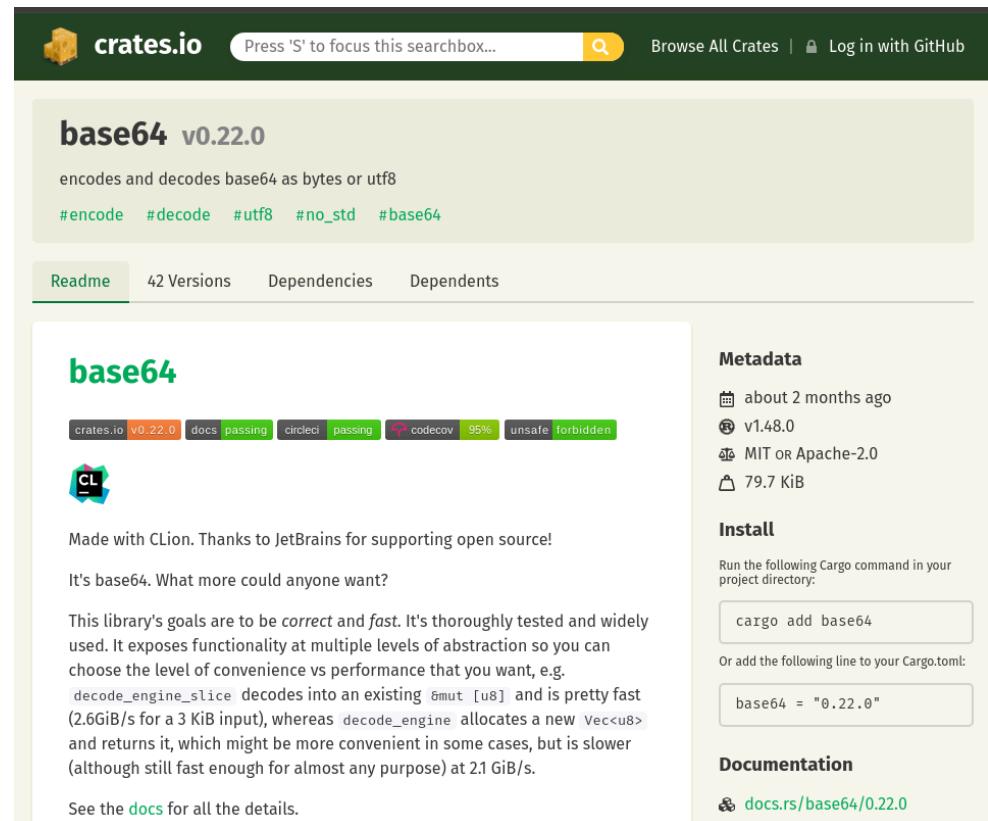
Let's build a very simple Rust binary and reverse it!

We will build a downloader, which will download and run a shell script.

(This is actually not that far off from real malware we're seeing.)

Terminology and tools : cargo

- **cargo**: The Rust build tool, package manager, and general “all in one developer tool”
 - Build your project
 - ⇒ e.g. “I want to build my binary, in release mode.”
 - Download new package dependencies for your project
 - ⇒ e.g. “I want to use the **base64** library, version 0.22.0, in my project”
- **rustc**: The actual Rust compiler.
 - Cargo runs **rustc**, plus a linker, to build your binary.
- Crate: Rust’s terminology for a package, or library
 - Central package repository for the Rust project: <https://crates.io/>



The screenshot shows the crates.io website for the **base64** crate. At the top, there's a navigation bar with the crates.io logo, a search bar containing "base64", and links for "Browse All Crates" and "Log in with GitHub". Below the header, the crate name **base64 v0.22.0** is displayed, along with a brief description: "encodes and decodes base64 as bytes or utf8". A code snippet shows dependencies: `#encode #decode #utf8 #no_std #base64`. Below this, there are tabs for "Readme" (which is selected), "42 Versions", "Dependencies", and "Dependents". The "Readme" section contains the title **base64**, a badge showing "crates.io v0.22.0", and a "CL" logo indicating support from JetBrains CLion. It also mentions "Made with CLion. Thanks to JetBrains for supporting open source!". The text describes the crate's goals: "correct and fast" and "thoroughly tested and widely used". It compares two decoding methods: `decode_engine_slice` (fast) and `decode_engine` (slower). Finally, it points to the [docs](#) for more details. To the right, there's a "Metadata" section with information about the crate's creation ("about 2 months ago"), version ("v1.48.0"), license ("MIT or Apache-2.0"), and size ("79.7 KiB"). There's also an "Install" section with the command `cargo add base64` and an "Add to Cargo.toml" section with the line `base64 = "0.22.0"`. The "Documentation" section links to [docs.rs/base64/0.22.0](#).

Setting up our first Rust program

```
cargo new simple-downloader  
cd simple-downloader  
  
$ cargo new simple-downloader  
    Created binary (application) `simple-downloader` package  
  
$ cd simple-downloader/  
  
$ tree  
.  
├── Cargo.toml  
└── src  
    └── main.rs
```

The default `main.rs`

```
1 fn main() {  
2     println!("Hello, world!");  
3 }
```

The default `cargo.toml` manifest file

```
1 [package]
2 name = "simple-downloader"
3 version = "0.1.0"
4 edition = "2021"
5
6 # See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html
7
8 [dependencies]
```

Notice how there's a lot missing here; a lot of the build configuration is based on default values, from [Cargo's built-in build profiles!](#)

Building our first Rust program

```
cd simple-downloader/  
cargo build  
  
$ cargo build  
Compiling simple-downloader v0.1.0 (/home/cxiao/presentations/rust-one-step-beyond-  
strings/simple-downloader)  
Finished dev [unoptimized + debuginfo] target(s) in 0.13s
```

With the defaults, note how this is an unoptimized build with full debugging information!

Cargo's default *dev* profile

 The Cargo Book > Cargo Reference > Profiles > Default Profiles

» Default profiles

dev

The `dev` profile is used for normal development and debugging. It is the default for build commands like `cargo build`, and is used for `cargo install --debug`.

The default settings for the `dev` profile are:

```
[profile.dev]
opt-level = 0
debug = true
split-debuginfo = '...' # Platform-specific.
strip = "none"
debug-assertions = true
overflow-checks = true
lto = false
panic = 'unwind'
incremental = true
codegen-units = 256
rpath = false
```



Running our first Rust program

```
cargo run  
$ cargo run  
    Finished dev [unoptimized + debuginfo] target(s) in 0.03s  
        Running `target/debug/simple-downloader`  
Hello, world!  
  
$ file target/debug/simple-downloader  
target/debug/simple-downloader: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),  
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,  
BuildID[sha1]=40da521231670b2f56459a8c981068ac28e57602, for GNU/Linux 3.2.0, with debug_info, not  
stripped
```

Notice how this is not stripped!

Examining target/debug/simple-downloader in our disassembler tool

[rust-one-step-beyond-strings] simple-downloader.bnbd — Binary Ninja 4.0.4958-Stable

File Edit View Analysis Debugger Plugins Window Help

Project Browser [rust-one-step-beyond-strings] simple-downloader.bnbd +

Symbols Q Search symbols ELF ▾ Linear ▾ High Level IL ▾

Name	Address
_init	0x0000000000
sub_6020	0x0000000000
sub_6036	0x0000000000
_Unwind_Resume	0x0000000000
sub_6046	0x0000000000
__cxa_finalize	0x0000000000
sub_6056	0x0000000000
core::str::pattern::simd_contains::[closure#2]	0x0000000000
core::slice::sort::break_patterns::hffaa4882e3480...	0x0000000000
core::slice::sort::partial_insertion_sort::hb88d72...	0x0000000000
core::slice::sort::heapsort::hd2622cb171c24b78	0x0000000000
core::panicking::assert_failed::hf08da4e67e51f1e	0x0000000000
alloc::raw_vec::{impl#2}::reserve::do_reserve_and...	0x0000000000
alloc::raw_vec::{impl#2}::reserve::do_reserve_and...	0x0000000000
alloc::raw_vec::{impl#2}::reserve::do_reserve_and...	0x0000000000
std::thread::ThreadId::new::exhausted::hc045eedee...	0x0000000000
std::io::buffered::BufWriter::BufWriter::write_all...	0x0000000000
std::process::process::abort::h1e35cb2fcc7af981	0x0000000000
std::sync::once::lock::OnceLock::initialize<std::sy...	0x0000000000
std::sync::once::lock::OnceLock::initialize<std::sy...	0x0000000000
std::sys::pal::common::small_c_string::run_with_cs...	0x0000000000
sub_7008	0x0000000000
std::sys::pal::common::small_c_string::run_with_cs...	0x0000000000
std::sys::pal::common::small_c_string::run_with_cs...	0x0000000000
sub_717d	0x0000000000
std::panicking::panic_count::is_zero_slow_path::hb...	0x0000000000
std::sys::pal::unix::locks::futex_mutex::Mutex::lo...	0x0000000000
std::sys::pal::unix::locks::futex_rwlock::RwLock::...	0x0000000000
std::sys::pal::unix::locks::futex_rwlock::RwLock::...	0x0000000000
std::sys_common::once::futex::Once::call::h18c5766...	0x0000000000
std::sys_common::once::futex::Once::call::h2e9646b...	0x0000000000
std::sys_common::once::futex::Once::call::h9610f2a...	0x0000000000
alloc::raw_vec::RawVec\$LT\$\$A\$GT\$::reserve::do_r...	0x0000000000
core::panicking::assert_failed::he7b9b8abdaeb96c...	0x0000000000
alloc::raw_vec::{impl#2}::reserve::do_reserve_and...	0x0000000000
alloc::alloc::handle_alloc_error::h836e982fea7018b...	0x0000000000
core::cell::panic_already_borrowed::ha88f2034c7bb6...	0x0000000000
core::option::unwrap_failed::hac39b9b7507453f8...	0x0000000000
core::option::expect_failed::h5c9b166b5a7a1f0...	0x0000000000
core::panicking::panic_fmt::hc69c4d258fe11477...	0x0000000000
core::panicking::panic_nounwind_fmt::h883e9894119f...	0x0000000000
core::panicking::panic::h90e84101c1877ef...	0x0000000000
core::panicking::panic_nounwind::h24432bd1d9d584b6...	0x0000000000
core::panicking::panic_nounwind_no_backtrace::hd62...	0x0000000000
core::panicking::panic_bounds_check::he5254f424ac3...	0x0000000000
core::panicking::panic_cannot_unwind::h7ede0171552...	0x0000000000
core::panicking::panic_in_cleanup::h7c6c61c84248d3...	0x0000000000
core::panicking::assert_failed_inner::hf6c64db802c7...	0x0000000000
core::result::unwrap_failed::hff299ec748d62aab...	0x0000000000
core::slice::index::slice_start_index_len_fail::hd...	0x0000000000
core::slice::index::slice_end_index_len_fail::h332...	0x0000000000
core::slice::index::slice_index_order_fail::hb034...	0x0000000000
core::slice::index::slice_end_index_overflow_fail...	0x0000000000
core::slice::index::copy_from_slice::len_mismatch...	0x0000000000
core::str::traits::str_index_overflow_fail::h889b9...	0x0000000000
core::str::slice_error_fail::hc482bbaa01e121c2...	0x0000000000
_start	0x0000000000
deregister_tm_clones	0x0000000000

```

void _start(int64_t arg1, int64_t arg2, void (* arg3)()) __noreturn
000087d5 core::panicking::panic_fmt::hc69c4d258fe11477()
000087d5 noreturn
000087db 0f 1f 44 00 00 ..D..
000087e0 void core::str::slice_error_fail::hc482bbaa01e121c2() __noreturn
000087e1 core::str::slice_error_fail_rt::hc84377fc0cf3c023()
000087e1 noreturn
000087e7 66-0f 1f 84 00 00 00 00 f.....
000087f0 void _start(int64_t arg1, int64_t arg2, void (* arg3)()) __noreturn
00008801 int64_t stack_end_1
00008801 int64_t stack_end = stack_end_1
0000880f __libc_start_main(main: main, argc: __return_addr.d, ubp_av: &ubp_av, init: nullptr, fini: nullptr, rtld_fini: arg3, stack_end: &stack_end)
0000880f noreturn
00008815 f4 66 2e-0f 1f 84 00 00 00 00 .f.....
00008820 void deregister_tm_clones()
00008848 | return
00008841 0f 1f 80 00 00 00 00
00008849 0f 1f 80 00 00 00 00 ..... .
00008850 void register_tm_clones()
00008888 | return
00008882 66 0f 1f 44 00 00 f..0..
00008889 0f 1f 80 00 00 00 00 ..... .
00008890 void _do_global_dtors_aux()
0000889b if (_TMC_END_ != 0)
0000889b | return
0000889b | if (_cxa_finalize != 0)
0000889b | | _cxa_finalize(d: &_dso_handle)
0000889b | deregister_tm_clones()
0000889c __TMC_END_ = 1
000088c5 0f 1f 00
000088c9 0f 1f 80 00 00 00 00 ..... .
000088d0 void frame_dummy()
000088d4 | return register_tm_clones() __tailcall
000088d9 0f 1f 80 00 00 00 00 ..... .
000088e0 struct std::process::ExitCode std::process::{impl#57}::report() __pure
000088e2 | return 0

```

▲ 4 linux-x86_64 0x880f-0x8815 (0x6 bytes)

Building our first Rust program, in release mode

```
cargo build --release
$ cargo build --release
    Compiling simple-downloader v0.1.0 (/home/cxiao/presentations/rust-one-step-beyond-
strings/simple-downloader)
        Finished release [optimized] target(s) in 0.20s

$ cargo run --release
    Finished release [optimized] target(s) in 0.00s
        Running `target/release/simple-downloader`
Hello, world!

$ file target/release/simple-downloader
target/release/simple-downloader: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=0e80062c94bf80a1e9e49fbceced23848f783031, for GNU/Linux 3.2.0, not stripped
```

Notice how this file still isn't stripped!

Examining target/release/simple-downloader in our disassembler tool

[rust-one-step-beyond-strings] simple-downloader.bnbd — Binary Ninja 4.0.4958-Stable

File Edit View Analysis Debugger Plugins Window Help

Project Browser [rust-one-step-beyond-strings] simple-downloader.bnbd +

Symbols Q Search symbols

ELF ▾ Linear ▾ High Level IL ▾

```

void _start(int64_t arg1, int64_t arg2, void (* arg3)()) __noreturn
    void** const var_30 = &data_56588
    int64_t var_28 = 1
    char const* const var_20 = "library/core/src/fmt/mod.rsfrom_str_radix_int: must lie in the r..."
    int128_t var_18 = zx.o(0)
    core::panicking::panic_fmt::hc69c4d258fe11477()
    noreturn
000087db      0f 1f 44 00 00      ..D..
000087e0      void core::str::slice_error_fail::hc482bbaa01e121c2(char* arg1, void* arg2, void* arg3, void* arg4) __noreturn
000087e1      core::str::slice_error_fail::hc84377fc0cf3c023(arg1, arg2, arg3, arg4)
000087e1      noreturn
000087e7      66-0f 1f 84 00 00 00 00      f.....
000087f0      void _start(int64_t arg1, int64_t arg2, void (* arg3)()) __noreturn
00008801      int64_t stack_end_1
00008801      int64_t stack_end = stack_end_1
0000880f      __libc_start_main(main: main, argc: __return_addr.d, ubp_av: &ubp_av, init: nullptr, fini: nullptr, rtld_fini: arg3, stack_end: &stack_end)
0000880f      noreturn
00008815      f4 66 2e-0f 1f 84 00 00 00 00      .f.....
00008820      void deregister_tm_clones()
00008848      return
00008841      0f 1f 80 00 00 00 00      .....
00008849      0f 1f 80 00 00 00 00      .....
00008850      void register_tm_clones()
00008888      return
00008882      66 0f 1f 44 00 00      f...
00008889      0f 1f 80 00 00 00 00      .....
00008890      void __do_global_dtors_aux()
0000889b      if (__TMC_END__ != 0)
000088c8      |   return
000088a9      if (__cxa_finalize != 0)
000088b2      |   __cxa_finalize(d: &__dso_handle)
000088b7      deregister_tm_clones()
000088bc      __TMC_END__ = 1
000088c5      0f 1f 00      ...
000088c9      0f 1f 80 00 00 00 00      .....
000088d0      void frame_dummy()
000088d4      return register_tm_clones() __tailcall
000088d9      0f 1f 80 00 00 00 00      .....

```

rust-one-step-beyond-strings

8 linux-x86_64 0x880f-0x8815 (0x6 bytes)

Cargo's default release profile

 The Cargo Book > Cargo Reference > Profiles > Default Profiles

Notice the use of `strip = "none"`

release

The `release` profile is intended for optimized artifacts used for releases and in production. This profile is used when the `--release` flag is used, and is the default for `cargo install`.

The default settings for the `release` profile are:

```
[profile.release]
opt-level = 3
debug = false
split-debuginfo = '...' # Platform-specific.
strip = "none"
debug-assertions = false
overflow-checks = false
lto = false
panic = 'unwind'
incremental = false
codegen-units = 16
rpath = false
```

Actually removing metadata

 The Cargo Book > Cargo Reference > Profiles > Profile Settings > strip

```
1 [package]
2 name = "simple-downloader"
3 version = "0.1.0"
4 edition = "2021"
5
6 # See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/man
7
8 [dependencies]
9
10 [profile.release]
11 strip = true # or strip = "symbols"
```

Building our Rust program, in release mode again

```
$ cargo build --release
   Compiling simple-downloader v0.1.0 (/home/cxiao/presentations/rust-one-step-beyond-
strings/simple-downloader)
     Finished release [optimized] target(s) in 0.22s

$ file target/release/simple-downloader
target/release/simple-downloader: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=8fa826275b714ff86a12498d3a9461e1eef61a3a, for GNU/Linux 3.2.0, stripped
```

[rust-one-step-beyond-strings] simple-downloader.bnbd — Binary Ninja 4.0.4958-Stable

File Edit View Analysis Debugger Plugins Window Help

← → Project Browser [rust-one-step-beyond-strings] simple-downloader.bnbd +

Symbols Search symbols ELF Linear High Level IL

Name	Address
_init	0x0000006000
sub_6020	0x0000006020
sub_6036	0x0000006036
_Unwind_Resume	0x0000006040
sub_6046	0x0000006046
_cxa_finalize	0x0000006050
sub_6056	0x0000006056
sub_6060	0x0000006060
sub_6190	0x0000006190
sub_6380	0x0000006380
sub_6830	0x0000006830
sub_6aa0	0x0000006aa0
sub_6ad0	0x0000006ad0
sub_6bb0	0x0000006bb0
sub_6c70	0x0000006c70
sub_6d50	0x0000006d50
sub_6d90	0x0000006d90
sub_6e10	0x0000006e10
sub_6e20	0x0000006e20
sub_6e60	0x0000006e60
sub_6eb0	0x0000006eb0
sub_7008	0x0000007008
sub_7030	0x0000007030
sub_70e0	0x00000070e0
sub_717d	0x000000717d
sub_71b0	0x00000071b0
sub_71d0	0x00000071d0
sub_72d0	0x00000072d0
sub_7430	0x0000007430
sub_7810	0x0000007510
sub_7740	0x0000007740
sub_7be0	0x0000007be0
sub_7de0	0x0000007de0
sub_7ea0	0x0000007ea0
sub_7ef0	0x0000007ef0
sub_7fb0	0x0000007fb0
sub_7fc0	0x0000007fc0
sub_8020	0x0000008020
sub_8040	0x0000008040
sub_80b0	0x00000080b0
sub_80f0	0x00000080f0
sub_8160	0x0000008160
sub_81b0	0x00000081b0
sub_8200	0x0000008200
sub_8250	0x0000008250
sub_82d0	0x00000082d0
sub_82f0	0x00000082f0
sub_8310	0x0000008310
sub_84d0	0x00000084d0
sub_8560	0x0000008560
sub_85e0	0x00000085e0
sub_8660	0x0000008660
sub_86e0	0x00000086e0
sub_8720	0x0000008720
sub_87a0	0x00000087a0
sub_87e0	0x00000087e0
_start	0x00000087f0
deregister_tm_clones	0x0000008820

```

void _start(int64_t arg1, int64_t arg2, void (* arg3)()) __noreturn
    0000874e    uint64_t (* var_38)(int64_t* arg1, int64_t* arg2) = sub_45110
    0000875a    void** const var_30 = &data_56658
    0000875f    int64_t var_28 = 3
    00008768    int64_t var_10 = 0
    00008776    int64_t** var_20 = &var_50
    0000877b    int64_t var_18 = 2
    0000878c    sub_80b0()
    0000878c    noreturn

    00008792    66 2e 0f 1f 84 00-00 00 00 0f 1f 40 00    f.....@.

    000087a0    void sub_87a0() __noreturn
    000087ae    void** const var_30 = &data_56588
    000087b3    int64_t var_28 = 1
    000087c3    char const* const var_20 = "library/core/src/fmt/mod.rsfrom_str_radix_int: must lie in the r..."
    000087cb    int128_t var_18 = zx.o(0)
    000087d5    sub_80b0()
    000087d5    noreturn

    000087db    0f 1f 44 00 00    ..D..

    000087e0    void sub_87e0(char* arg1, void* arg2, void* arg3, void* arg4) __noreturn
    000087e1    sub_4100(arg1, arg2, arg3, arg4)
    000087e1    noreturn

    000087e7    66-0f 1f 84 00 00 00 00 00    f.....
    000087f0    void _start(int64_t arg1, int64_t arg2, void (* arg3)()) __noreturn
    00008801    int64_t stack_end_1
    00008801    int64_t stack_end = stack_end_1
    0000880f    libc_start_main(main: main, argc: __return_addr.d, ubp_av: &ubp_av, init: nullptr, fini: nullptr, rtld_fini: arg3, stack_end: &stack
    0000880f    noreturn

    00008815    f4 66 2e-0f 1f 84 00 00 00 00 00    .f.....
    00008820    void deregister_tm_clones()
    00008848    return

    00008841    0f 1f 80 00 00 00 00
    00008849    0f 1f 80 00 00 00 00    .....
    00008850    void sub_8850()
    00008888    return

    00008882    66 0f 1f 44 00 00    f..0..
    00008889    0f 1f 80 00 00 00 00    .....
    00008890    void sub_8890()
    0000889b    if (data_57058 != 0)
    000088c8    return
    000088a9    if (_cxa_finalize != 0)
```

rust-one-step-beyond-strings

linux-x86_64 0x880f-0x8815 (0x6 bytes)

Adding a library



 crates.io Press 'S' to focus this searchbox... 

Browse All Crates |  Log in with GitHub

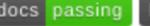
reqwest v0.12.2

higher level HTTP client library

#client #http #request

[Readme](#) [93 Versions](#) [Dependencies](#) [Dependents](#)

reqwest

An ergonomic, batteries-included HTTP Client for Rust.

- Async and blocking clients
- Plain bodies, JSON, urlencoded, multipart
- Customizable redirect policy
- HTTP Proxies
- HTTPS via system-native TLS (or optionally, rustls)
- Cookie Store
- WASM

Example

Metadata

 3 days ago
 v1.63.0
 MIT OR Apache-2.0
 163 KiB

Install

Run the following Cargo command in your project directory:

```
cargo add reqwest
```

Or add the following line to your Cargo.toml:

```
reqwest = "0.12.2"
```

Adding the reqwest library

```
cargo add reqwest --features blocking  
$ cargo add reqwest --features blocking  
  Updating crates.io index  
    Adding reqwest v0.12.4 to dependencies.  
      Features:  
        + __tls  
        + blocking  
        + charset  
        + default-tls  
        + h2  
        + http2  
        + macos-system-configuration  
        25 deactivated features  
  Updating crates.io index
```

Updated `cargo.toml` with the `reqwest` library

```
1 [package]
2 name = "simple-downloader"
3 version = "0.1.0"
4 edition = "2021"
5
6 # See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/man
7
8 [profile.release]
9 strip = true # or strip = "symbols"
10
11 [dependencies]
12 reqwest = { version = "0.12.2", features = ["blocking"] }
```

Writing the download functionality



Literally copying the example from the `reqwest` documentation

```
1 fn main() {  
2     let request_result = reqwest::blocking::get("https://sh.rustup.rs/rustup-init.sh")  
3     match request_result {  
4         Ok(response) => { // Use the Ok variant of the Result enum, which contains a  
5             println!("Received response: {response:#?}");  
6         },  
7         Err(error) => { // Use the Err variant of the Result enum, which contains an  
8             panic!("Error making request: {error}");  
9         },  
10    }  
11 }
```

The screenshot shows a Rust IDE interface with a search bar at the top containing "simple-downloader". Below the search bar, there are tabs for "main.rs 3" and "src/main.rs/ main". A code editor window displays the following Rust code:

```
1 fn main() {  
2     let request_result = request::blocking::get(Q)  
3 }  
4
```

A tooltip is displayed over the code editor, providing documentation for the `get` function:

fn get(url: T) → Result<Response, Error>
Shortcut method to quickly make a *blocking* GET request.
NOTE: This function creates a new internal Client on each call, and so should not be used if making many requests. Create a Client instead.

Below the code editor, the status bar shows:

< master* ⚡ ⚡ 3 ⚡ 0 ⚡ 0 rust-analyzer -- INSERT --

Ln 2, Col 49 Spaces: 4 UTF-8 LF Rust

Examples

```
let body =  
    request::blocking::get("https://www.rust-lang.org")?
```

The screenshot shows a code editor window for a Rust project named "simple-downloader". The file being edited is "src/main.rs". The code in the editor is:

```
fn main() {  
    let request_result = reqwest::blocking::get(url: "https://rust-lang.org/");  
}
```

A tooltip is displayed over the line "let request_result = reqwest::blocking::get(url: "https://rust-lang.org/");". The tooltip content is:

unused variable: `request_result`
`#[warn(unused_variables)]` on by default rustc([Click for full compiler diagnostic](#))
main.rs(2, 9): if this is intentional, prefix it with an underscore:
`_request_result`
// size = 176 (0xB0), align = 0x8
let request_result: Result<Response, Error>

Go to [Result](#) | [Response](#) | [Error](#)
[View Problem \(Alt+F8\)](#) [Quick Fix... \(Ctrl+.\)](#)

The status bar at the bottom of the IDE shows the following information:

- master* (branch)
- 0 ▲ 1 (diff count)
- 0 (warning count)
- rust-analyzer (language server)
- "main.rs" 4L 89C written (file statistics)
- Ln 2, Col 22 (cursor position)
- Spaces: 4 (text encoding)
- UTF-8 (character encoding)
- LF (line ending)
- Rust (language)
- Bell icon (notifications)

The screenshot shows a code editor window for a Rust project named "simple-downloader". The file being edited is "src/main.rs". A tooltip is displayed over the code at line 3, column 24, highlighting a missing match arm in a match expression. The tooltip text reads: "missing match arm: `Ok(_)` and `Err(_)` not covered rust-analyzer(E0004)". Below the tooltip, there are links to "View Problem (Alt+F8)" and "Quick Fix... (Ctrl+.)".

```
fn main() {  
    let request_result = reqwest::blocking::get(url: "https://rust-lang.org/");  
    match request_result {}  
}
```

The screenshot shows a code editor window for a Rust project named "simple-downloader". The file being edited is "src/main.rs". A tooltip is displayed over the variable "_response" at line 2, column 15. The tooltip contains the following information:

- // Implements notable traits: Read
- // size = 176 (0xB0), align = 0x8
- let _response: Response
- (url: "https://rust-lang.org/");

The tooltip also includes links to "Go to Response" and "Read". The main code in "main.rs" is as follows:

```
fn main() {  
    let req = Client::new().get("https://rust-lang.org/").send();  
    match req.  
        Ok(_response) => {}, // Use the Ok variant of the Result enum, which contains a Response  
        Err(_error) => {}, // Use the Err variant of the Result enum, which contains an Error  
    }  
}
```

The status bar at the bottom of the editor shows the following information:

- master* (branch)
- 0 ▲ 0 (diff count)
- 0 (warning count)
- rust-analyzer (language server)
- "main.rs" 8L 313C written (file details)
- Ln 4, Col 20 (cursor position)
- Spaces: 4 (text encoding)
- UTF-8 (text encoding)
- LF (text encoding)
- Rust (language)
- Unread notifications (bell icon)

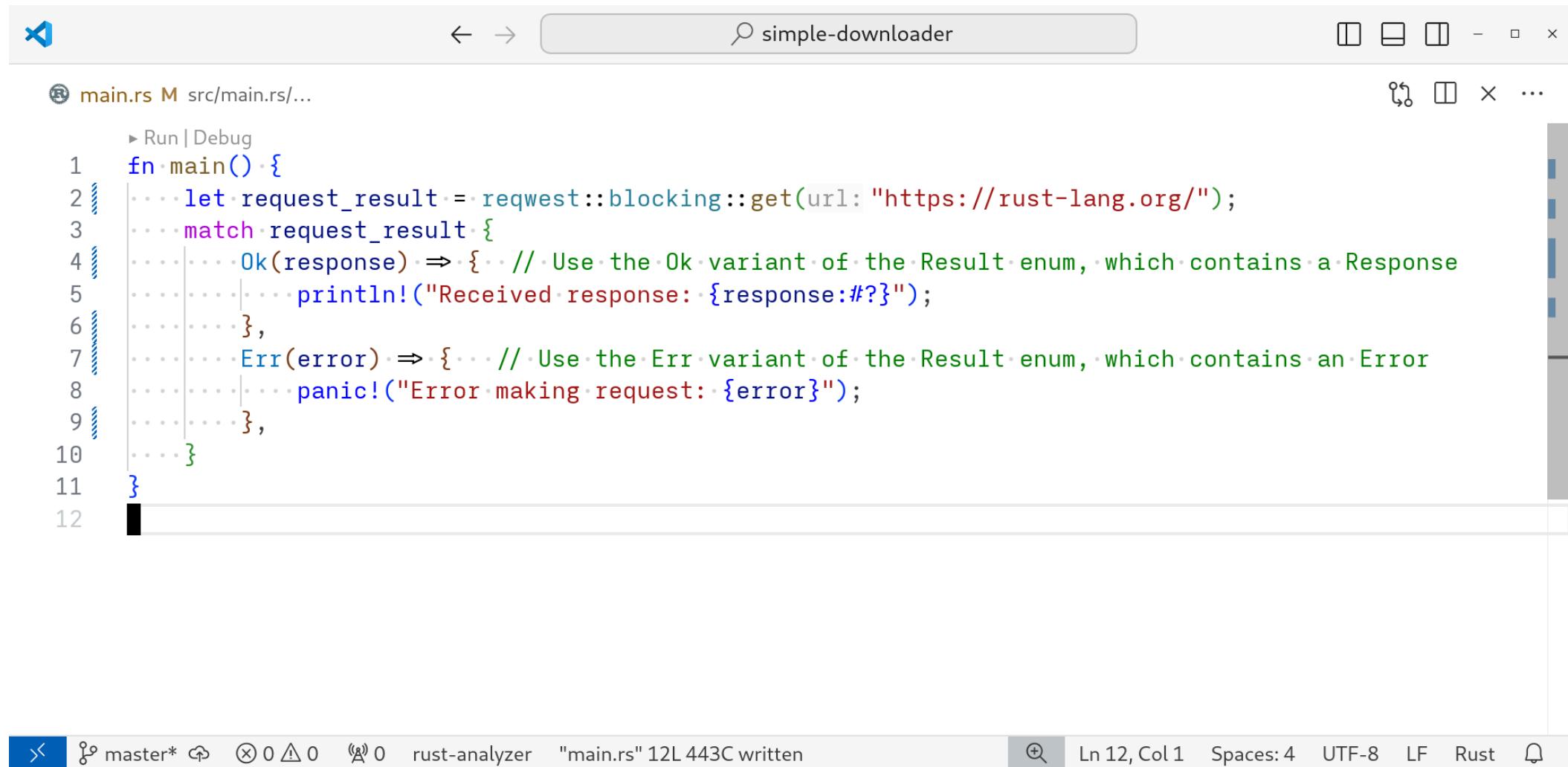
The screenshot shows a Rust IDE interface with the project name "simple-downloader" in the search bar. The main editor window displays the following Rust code:

```
fn main() {  
    let request_result = reqwest::blocking::get(url: "https://rust-lang.org/");  
    match request_result {  
        Ok(response) => { // Use the Ok variant of the Result enum, which contains a Response  
            println!("Received response: {response:#?}");  
        },  
        Err(e) {  
            std::macros::macro_rules! println // matched arm #1  
                Prints to the standard output, with a newline.  
                On all platforms, the newline is the LINE FEED character ( \n / U+000A ) alone (no additional CARRIAGE RETURN ( \r / U+000D )).  
                This macro uses the same syntax as format, but writes to the standard output instead. See [ std::fmt ] for more information.  
                The println! macro will lock the standard output on each call. If you call println! within a hot loop, this behavior may be the bottleneck of the loop. To avoid this, lock stdout with io::stdout().lock :  
        }  
    }  
}
```

A tooltip is displayed over the `Err` branch, providing documentation for the `println!` macro. The tooltip content is as follows:

- Prints to the standard output, with a newline.
- On all platforms, the newline is the LINE FEED character (\n / U+000A) alone (no additional CARRIAGE RETURN (\r / U+000D)).
- This macro uses the same syntax as `format`, but writes to the standard output instead. See [`std::fmt`] for more information.
- The `println!` macro will lock the standard output on each call. If you call `println!` within a hot loop, this behavior may be the bottleneck of the loop. To avoid this, lock stdout with `io::stdout().lock` :

The status bar at the bottom shows the current file is "master*", has 1 warning, and is using the "rust-analyzer" language server. It also indicates the code is in "NORMAL" mode, has 4 spaces, and is using UTF-8 encoding.



simple-downloader

```
main.rs M src/main.rs/... ▶ Run | Debug
1 fn main() {
2     let request_result = reqwest::blocking::get(url: "https://rust-lang.org/");
3     match request_result {
4         Ok(response) => { // Use the Ok variant of the Result enum, which contains a Response
5             println!("Received response: {response:#?}");
6         },
7         Err(error) => { // Use the Err variant of the Result enum, which contains an Error
8             panic!("Error making request: {error}");
9         },
10    }
11 }
12
```

master* 0 0 0 rust-analyzer "main.rs" 12L 443C written

Ln 12, Col 1 Spaces: 4 UTF-8 LF Rust

Building the binary, with just the download functionality

```
cargo build --release
```

The first time we build this after adding our **reqwest** dependency, we see that all of **reqwest**, and *its* dependencies, are being built:

```
$ cargo build --release
Compiling libc v0.2.153
Compiling proc-macro2 v1.0.79
Compiling unicode-ident v1.0.12
Compiling pin-project-lite v0.2.13
Compiling once_cell v1.19.0
Compiling bytes v1.6.0
Compiling pkg-config v0.3.30
Compiling vcpkg v0.2.15
Compiling futures-core v0.3.30
Compiling log v0.4.21
Compiling cc v1.0.90
Compiling autocfg v1.2.0
Compiling itoa v1.0.11
Compiling futures-sink v0.3.30
Compiling tracing-core v0.1.32
Compiling fnv v1.0.7
Compiling slab v0.4.9
Compiling http v1.1.0
```

 crates.io Press 'S' to focus this searchbox...  Browse All Crates |  Log in with GitHub

reqwest v0.12.2

higher level HTTP client library

#client #http #request

Readme 93 Versions Dependencies Dependents

Dependencies

^0.21	base64 encodes and decodes base64 as bytes or utf8
^1.0	bytes Types and traits for working with bytes
^0.3.0	futures-core NO DEFAULT FEATURES The core traits and types in for the `futures` library.

Running the binary, with just the download functionality

```
cargo run --release  
$ cargo run --release  
    Finished release [optimized] target(s) in 0.09s  
    Running `target/release/simple-downloader`  
Received response: Response {  
    url: Url {  
        scheme: "https",  
        cannot_be_a_base: false,  
        username: "",  
        password: None,  
        host: Some(  
            Domain(  
                "www.rust-lang.org",  
            ),  
        ),  
        port: None,  
        path: "/",  
        query: None,  
        fragment: None,  
    },  
},
```

Reversing our binary, with just the download functionality

Exercise A: Initial string triage

What happens if we just run **strings** on our binary?

```
0x1f5000 | /home/cxiao/.cargo/registry/src/index.crates.io-6f17d22bba15001f/tokio-  
1.37.0/src-sync/mpsc/list.rshttps://www.rust-lang.orgReceived response:  
0x1f5090 | Error making request:  
0x1f50b8 | failed to write whole  
buffer/rustc/7cf61ebde7b22796c69757901dd346d0fe70bd97/library/std/src/io/mod.rsformatter  
errorfatal runtime error: thread result panicked on drop  
0x1f51d8 | /rustc/7cf61ebde7b22796c69757901dd346d0fe70bd97/library/core/src/sync/atomic.rsthere is  
no such thing as a release failure orderingthere is no such thing as an acquire-release failure  
ordering  
0x1f52a0 | GGGGGGGGGGGGGGGInactiveStreamIdMalformedHeadersinvalid ping  
frame/home/cxiao/.cargo/registry/src/index.crates.io-6f17d22bba15001f/h2-  
0.4.3/src/proto/ping_pong.rsinvalid pong frame  
0x1f5358 | Arc counter  
overflow/rustc/7cf61ebde7b22796c69757901dd346d0fe70bd97/library/alloc/src/sync.rs/home/cxiao/.cargo  
6f17d22bba15001f/tokio-1.37.0/src/sync/oneshot.rscannot access a Thread Local Storage value during  
or after  
destruction/rustc/7cf61ebde7b22796c69757901dd346d0fe70bd97/library/std/src/thread/local.rsCannot  
start a runtime from within a runtime. This happens because a function (like `block_on`) attempted  
to block the current thread while the thread is being used to drive asynchronous tasks.failed to
```

Breaking down the strings

Rust strings are generally not null-terminated. All of this is from a single null-terminated string at **0x1f5358**:

```
Arc counter overflow
/rustc/7cf61ebde7b22796c69757901dd346d0fe70bd97/library/alloc/src/sync.rs
/home/cxiao/.cargo/registry/src/index.crates.io-6f17d22bba15001f/tokio-1.37.0/src/sync/oneshot.rs
cannot access a Thread Local Storage value during or after destruction
/rustc/7cf61ebde7b22796c69757901dd346d0fe70bd97/library/std/src/thread/local.rs
Cannot start a runtime from within a runtime. This happens because a function (like `block_on`)
attempted to block the current thread while the thread is being used to drive asynchronous tasks.
failed to park thread
/home/cxiao/.cargo/registry/src/index.crates.io-6f17d22bba15001f/tokio-
1.37.0/src/runtime/scheduler/multi_thread/mod.rs
/home/cxiao/.cargo/registry/src/index.crates.io-6f17d22bba15001f/tokio-
1.37.0/src/runtime/scheduler/current_thread/mod.rs
Failed to `Enter::block_on`
UnexpectedFrameTypePayloadTooBigRejectedReleaseCapacityTooBigOverflowedStreamIdMissingUriSchemeAndA
called after complete
```

Breaking down the strings

What metadata can we get from this?

- The version of Rust this binary was compiled with:

/rustc/7cf61ebde7b22796c69757901dd346d0fe70bd97/library/alloc/src/sync.rs

→ Specifically, this is the Git commit hash of the main Rust language Git repository.

→  [github.com/rust-](https://github.com/rust-lang/rust/blob/7cf61ebde7b22796c69757901dd346d0fe70bd97/library/alloc/src/sync.rs)

[lang/rust/blob/7cf61ebde7b22796c69757901dd346d0fe70bd97/library/alloc/src/sync.rs](https://github.com/rust-lang/rust/blob/7cf61ebde7b22796c69757901dd346d0fe70bd97/library/alloc/src/sync.rs)

 **library/alloc/src/sync.rs** is a source file that is part of the *standard library*. **rustc**, however, is the Rust compiler. Why are these both here?

The Language, the Compiler, and the Standard Library

The Rust compiler and standard library are in the same repository and are released together.

Therefore, when we talk about “Rust version 1.77.1”, we mean both

- “Rust compiler (`rustc`) version 1.77.1”, and
- “Rust standard library version 1.77.1”

1.76.0

- Released on: *8 February, 2024*
- Branched from master on: *22 December, 2023*

Language

- Document Rust ABI compatibility between various types
- Also: guarantee that `char` and `u32` are ABI-compatible
- Add lint `ambiguous_wide_pointer_comparisons` that supersedes `clippy::vtable_address_comparisons`

Compiler

- Lint pinned `#[must_use]` pointers (in particular, `Box<T>` where `T` is `#[must_use]`) in `unused_must_use`.
- Soundness fix: fix computing the offset of an unsized field in a packed struct
- Soundness fix: fix dynamic size/align computation logic for packed types with dyn Trait tail
- Add `$message_type` field to distinguish json diagnostic outputs
- Enable Rust to use the EHCont security feature of Windows
- Add tier 3 `{x86_64,i686}-win7-windows-msvc` targets
- Add tier 3 `aarch64-apple-watchos` target
- Add tier 3 `arm64e-apple-ios & arm64e-apple-darwin` targets

Refer to Rust’s [platform support page][platform-support-doc] for more information on Rust’s tiered platform support.

Libraries

- Add a column number to `dbg!()`
- Add `std::hash::{DefaultHasher, RandomState}` exports
- Fix rounding issue with exponents in `fmt`
- Add `T: ?Sized` to `RwLockReadGuard` and `RwLockWriteGuard`’s `Debug` impls.
- Windows: Allow `File::create` to work on hidden files

<https://releases.rs/docs/1.76.0>

Breaking down the strings

- The Cargo home path of the machine where the binary was built:
`/home/cxiao/.cargo/`
- Some, but not all of the dependency tree for the binary:
`/home/cxiao/.cargo/registry/src/index.crates.io-6f17d22bba15001f/tokio-1.37.0/`
 - Why not the entire dependency tree? We will see soon...

Metadata for this binary, with *rustbininfo*



github.com/N0fix/rustbininfo

```
$ rustbininfo info ~/presentations/rust-one-step-beyond-strings/samples/simple-downloader-release-build-stripped
```

```
TargetRustInfo(  
    rustc_version='1.77.1',  
    rustc_commit_hash='7cf61ebde7b22796c69757901dd346d0fe70bd97',  
    dependencies=[  
        Crate(name='base64', version='0.21.7', features=[], repository=None),  
        Crate(name='bytes', version='1.6.0', features=[], repository=None),  
        Crate(name='encoding_rs', version='0.8.33', features=[], repository=None),  
        Crate(name='futures-channel', version='0.3.30', features=[], repository=None),  
        Crate(name='futures-core', version='0.3.30', features=[], repository=None),  
        Crate(name='futures-util', version='0.3.30', features=[], repository=None),  
        Crate(name='h2', version='0.4.3', features=[], repository=None),  
        Crate(name='hashbrown', version='0.14.3', features=[], repository=None),  
        Crate(name='http', version='1.1.0', features=[], repository=None),  
        Crate(name='http-body-util', version='0.1.1', features=[], repository=None),  
        Crate(name='httparse', version='1.8.0', features=[], repository=None),  
        Crate(name='hyper', version='1.2.0', features=[], repository=None),  
        Crate(name='hyper-tls', version='0.6.0', features=[], repository=None),  
        Crate(name='hyper-util', version='0.1.3', features=[], repository=None),  
        Crate(name='idna', version='0.5.0', features=[], repository=None),
```

Where do these strings come from? The concept of panic

Unrecoverable, unexpected errors that cannot be caught or handled.

- The standard behaviour is that the current thread exits (after calling destructors)
- Can be added by the programmer (like we did in our program)
- Also lots of places in the standard library where it can occur:
 - Bounds check failures at runtime
 - Assertion failures
 - Malformed or unexpected state



In fact, there are some good examples of these conditions in the strings we just looked at!

Why is this here? Where does this information show up?

Let's trigger the code path in our binary where the request fails, and the panic occurs.

```
thread 'main' panicked at src/main.rs:8:13:  
Error making request: error sending request for url (https://localhost:1234/)  
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

@@ Interesting that we have a line and column number there...

Looking at the panic! in our unstripped binary

```

1 // ! This is a Ghidra decompilation, not real source code!
2 void simple_downloader::main(void)
3 {
4     reqwest::blocking::get(local_e0,"https://sh.rustup.rs/rustup-init.sh",0x23);
5     if (local_e0[0] == 3) { // ! ← Handle the state where the request failed
6         local_190[0] = local_d8;
7         local_258 = _<>::fmt;
8         local_330 = &PTR_s_Error_making_request:_003fd6d0;
9         local_328 = (undefined ***)0x1;
10        pvStack_310 = (void *)0x0;
11        pppuStack_320 = (undefined ****)&local_260;
12        local_318 = 1;
13        local_260 = local_190;
14        // ! ↓ Where our actual panic occurs,
15        // and where the printed panic message is constructed.
16        core::panicking::panic_fmt(&local_330,&PTR_s_src/main.rs_003fd6e0);
17    }

```

Looking into the guts of panic

```

1 pub const fn core::panicking::panic_fmt(fmt: fmt::Arguments<'_>) -> ! {
2     if cfg!(feature = "panic_immediate_abort") {
3         super::intrinsics::abort()
4     }
5
6     // NOTE This function never crosses the FFI boundary; it's a Rust-to-Rust call
7     // that gets resolved to the `#[panic_handler]` function.
8     extern "Rust" {
9         #[lang = "panic_impl"]
10        fn panic_impl(pi: &PanicInfo<'_>) -> !;
11    }
12
13    // ⚠️ This is our function call that we see in the decompilation!
14    let pi = PanicInfo::internal_constructor(
15        Some(&fmt),
16        Location::caller(), // ⚠️ Returns a `core::panic::Location`
17        /* can_unwind */ true,
18        /* force_no_backtrace */ false,
19    );
20
21    // SAFETY: `panic_impl` is defined in safe Rust code and thus is safe to call.
22    unsafe { panic_impl(&pi) }
23 }
```

- 📖 Docs: doc.rust-lang.org/core/panicking/fn.panic_fmt.html
- 🟡 Source: github.com/rust-lang/rust/blob/1.77.1/library/core/src/panicking.rs#L52

Looking into the guts of panic

In practice, compiler magic just replaces this reference to the actual `core::panic::Location` in the binary.

```
1 impl<'a> Location<'a> {
2     pub const fn caller() -> &'static Location<'static> {
3         crate::intrinsics::caller_location()
4     }
5 }
```

- 📄 Docs: stdrs.dev/nightly/x86_64-unknown-linux-gnu/core/panic/location/struct.Location.html#method.caller
- 💡 Source: <https://github.com/rust-lang/rust/blob/1.77.1/library/core/src/panic/location.rs#L87>

The `core::panic::Location` metadata struct

```
1 pub struct core::panic::Location<'a> {  
2     file: &'a str,  
3     line: u32,  
4     col: u32,  
5 }
```

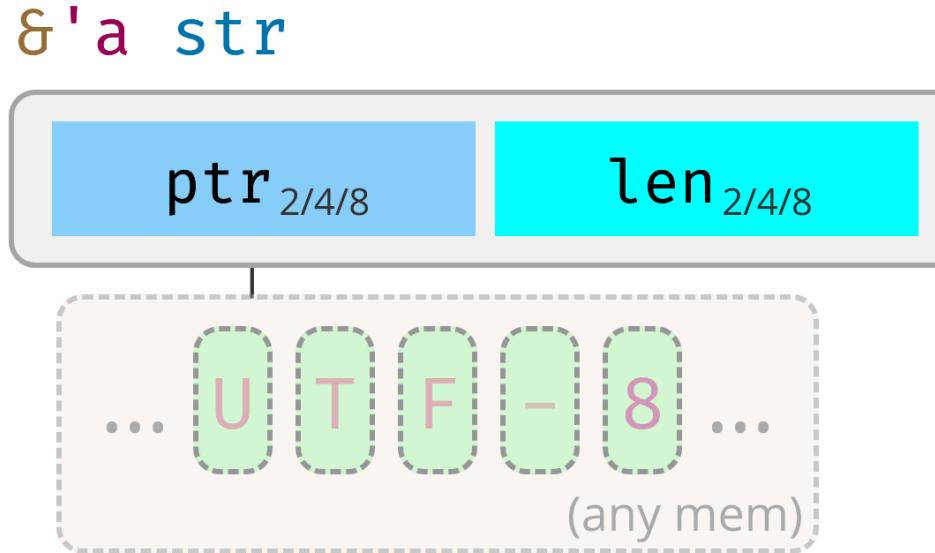
- 📄 Docs: stdrs.dev/nightly/x86_64-unknown-linux-gnu/core/panic/struct.Location.html
- ⚡ Source: github.com/rust-lang/rust/blob/1.77.1/library/core/src/panic/location.rs#L33

⚠ Caution: Type layouts in compiled Rust binaries [are not stable!](#)

- You must do the work of examining the data in your particular binary, and deducing from that what the layout of `core::panic::Location` in your particular binary is!

Type layouts in memory: The “pointer to constant string” type, `&str`

-  Docs: doc.rust-lang.org/std/primitive.str.html
-  Cheat Sheet: cheats.rs/#pointer-meta



String slice reference (i.e., the reference type of string type `str`), with meta `len` being byte length.

Putting it all together: `core::panic::Location` structs in our binary

	PTR_s_src/main.rs_003fd6e0		XREF[1]: main:00148ea9(*)	
	addr	s_src/main.rs_0031118a		= "src/main.rs"
003fd6e0	8a 11 31 00 00 00 00 00	??	0Bh	
003fd6e8	0b	??	00h	
003fd6e9	00	??	00h	
003fd6ea	00	??	00h	
003fd6eb	00	??	00h	
003fd6ec	00	??	00h	
003fd6ed	00	??	00h	
003fd6ee	00	??	00h	
003fd6ef	00	??	00h	
003fd6f0	1c	??	1Ch	
003fd6f1	00	??	00h	
003fd6f2	00	??	00h	
003fd6f3	00	??	00h	
003fd6f4	0d	??	0Dh	
003fd6f5	00	??	00h	
003fd6f6	00	??	00h	
003fd6f7	00	??	00h	

Putting it all together: `core::panic::Location` structs in our binary

		core::panic::Location_003fd6e0	XREF[1]:	main:00148ea9(*)
⊕	003fd6e0	8a 11 31 core::panic::Location 00 00 00 00 00 0b ...		
	003fd6e0	8a 11 31 00 00 char * s_src/main.rs_0031118a str_ptr 00 00 00	= "src/main.rs"	XREF[1]:
	003fd6e8	0b 00 00 00 00 uint64_t Bh str_len 00 00 00		
	003fd6f0	1c 00 00 00 uint32_t 1Ch line 0d 00 00 00 uint32_t Dh col		

Using the *panic* metadata to our advantage

🔗 Using panic metadata to recover source code information from Rust binaries - cxiao.net

Tags	Location	Content	Preview
Tags	18000db2d	... library\std\src\sys\windows\path.rs: line 104, col 35	4989c0 mov r8, rax
	18000db3c	... library\std\src\sys\windows\path.rs: line 104, col 35	4989c0 mov r8, rax
	18000db46	... library\std\src\sys\windows\path.rs: line 96, col 10	4989c0 mov r8, rax
	18000db55	... library\std\src\sys\windows\path.rs: line 96, col 10	4989c0 mov r8, rax
	18000db5f	... library\std\src\sys\windows\path.rs: line 104, col 35	4989c0 mov r8, rax
	18000db6e	... library\std\src\sys\windows\path.rs: line 104, col 35	4989c0 mov r8, rax
	18000db78	... library\std\src\sys\windows\path.rs: line 90, col 9	4989c0 mov r8, rax
	18000db8e	... library\std\src\sys\windows\path.rs: line 104, col 35	4989c0 mov r8, rax
	18000db9d	... library\std\src\sys\windows\path.rs: line 104, col 35	4989c0 mov r8, rax
		library\std\src\sys\windows\stdio.rs - Rust Panic Lo...	
	1800052e0	... library\std\src\sys\windows\stdio.rs: line 165, col 60	488d0549590300 lea ...
	1800052e7	... library\std\src\sys\windows\stdio.rs: line 165, col 60	4889442420 mov qword...
	18000dd00	... library\std\src\sys\windows\stdio.rs: line 165, col 35	4889442420 mov qword...
	18000dd07	... library\std\src\sys\windows\stdio.rs: line 165, col 35	4889442420 mov qword...
	18000dd14	... library\std\src\sys\windows\stdio.rs: line 165, col 35	e8576affff call sub...
	18000de4b	... library\std\src\sys\windows\stdio.rs: line 106, col 9	488d15fecb0200 lea ...
	18000de5d	... library\std\src\sys\windows\stdio.rs: line 125, col 33	4989c0 mov r8, rax
	18000de6c	... library\std\src\sys\windows\stdio.rs: line 125, col 33	4989c0 mov r8, rax
	18000de7e	... library\std\src\sys\windows\stdio.rs: line 129, col 17	4c8d0d63cd0200 lea ...
	18000de93	... library\std\src\sys\windows\stdio.rs: line 131, col 17	4c8d0d66cd0200 lea ...
	18000e0c1	... library\std\src\sys\windows\stdio.rs: line 205, col 41	
	18000e0fe	... library\std\src\sys\windows\stdio.rs: line 188, col 9	488d157bcb0200 lea ...
	18000e111	... library\std\src\sys\windows\stdio.rs: line 191, col 45	
	18000e127	... library\std\src\sys\windows\stdio.rs: line 214, col 19	
		library\std\src\sys\windows\thread_local_key.rs - R...	
	18000e4da	... library\std\src\sys\windows\thread_local_key.rs: lin...	4c8d052fc80200 lea ...

Limits of relying on panic metadata

The metadata doesn't have to be there, and developers can remove it:

```
RUSTFLAGS="-Zlocation-detail=none" cargo +nightly build --release
```

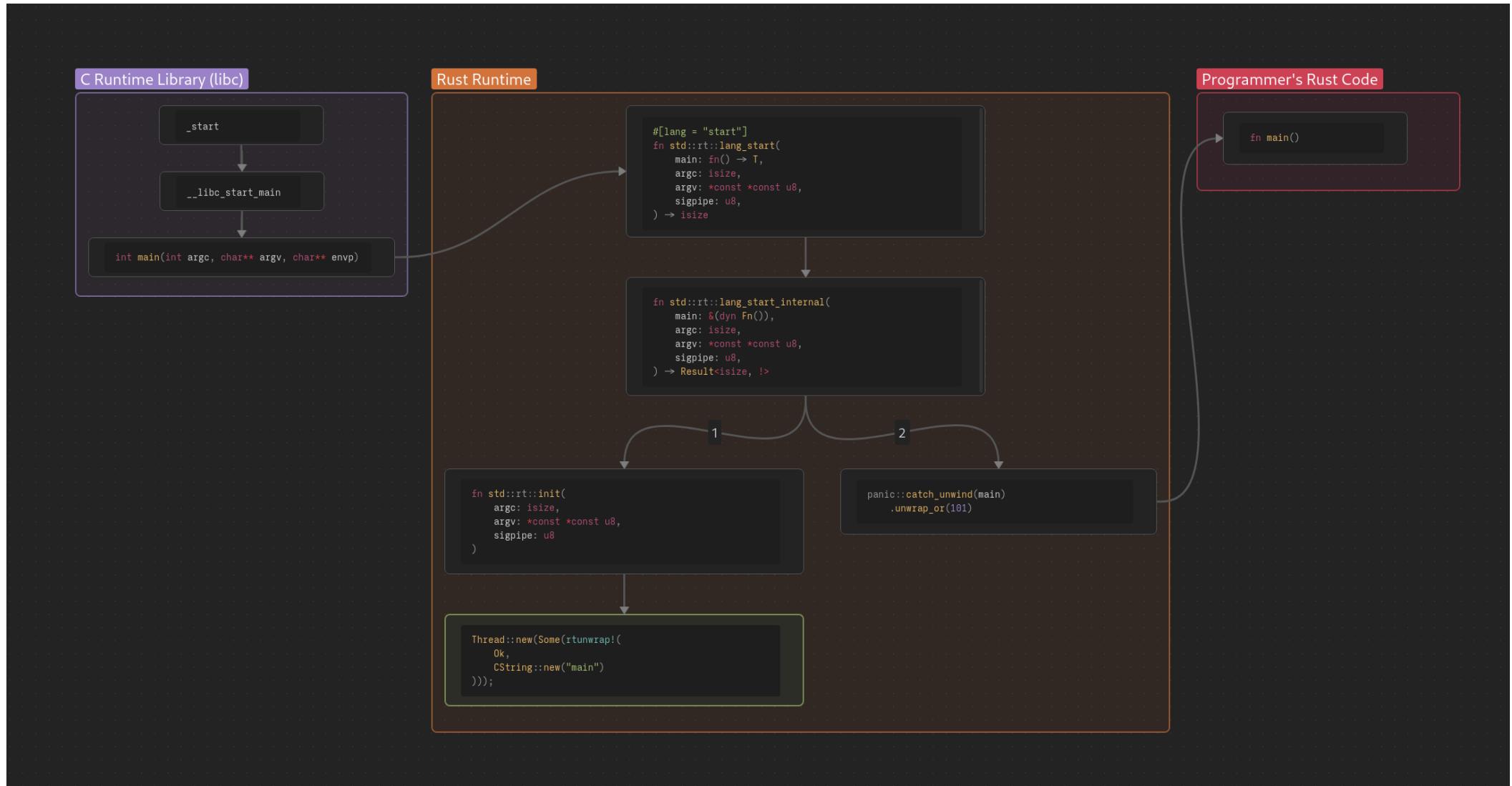
Most Rust malware we're seeing still has this information intact, though.

In general, this is a good example of how understanding the language features *helps us* as reversers.

Exercise B: Finding the entry point

We're going to switch over to our stripped binary now, and find its entry point.

A quick look at the runtime initialization



 github.com/rust-lang/rust/blob/1.77.1/library/std/src/rt.rs#L159

```
1  #[cfg(not(any(test, doctest)))]
2  #[lang = "start"]
3  fn lang_start<T: crate::process::Termination + 'static>(
4      main: fn() -> T,
5      argc: isize,
6      argv: *const *const u8,
7      sigpipe: u8,
8  ) -> isize {
9      let Ok(v) = lang_start_internal(
10          &move || crate::sys_common::backtrace::__rust_begin_short_backtrace(main).repo
11          argc,
12          argv,
13          sigpipe,
14      );
15      v
16 }
```

 github.com/rust-lang/rust/blob/1.77.1/library/std/src/rt.rs#L125

```
1 // To reduce the generated code of the new `lang_start`, this function is doing
2 // the real work.
3 #[cfg(not(test))]
4 fn lang_start_internal(
5     main: &(dyn Fn() -> i32 + Sync + crate::panic::RefUnwindSafe),
6     argc: isize,
7     argv: *const *const u8,
8     sigpipe: u8,
9 ) -> Result<isize, !> {
10     use crate::{mem, panic};
11     let rt_abort = move |e| {
12         mem::forget(e);
13         rtabort!("initialization or cleanup bug");
14     };
15     // Guard against the code called by this function from unwinding outside of the Ru
16     // code, which is UB. This is a requirement imposed by a combination of how the
17     // `#[lang="start"]` attribute is implemented as well as by the implementation of
18     // mechanism itself.
19     //
```

 github.com/rust-lang/rust/blob/1.77.1/library/std/src/rt.rs#L104

```
1 unsafe fn init(argc: isize, argv: *const *const u8, sigpipe: u8) {
2     unsafe {
3         sys::init(argc, argv, sigpipe);
4
5         let main_guard = sys::thread::guard::init();
6         // Next, set up the current Thread with the guard information we just
7         // created. Note that this isn't necessary in general for new threads,
8         // but we just do this to name the main thread and to give it correct
9         // info about the stack bounds.
10        let thread = Thread::new(Some(rtunwrap!(Ok, CString::new("main"))));
11        thread_info::set(main_guard, thread);
12    }
13 }
```



github.com/rust-lang/rust/blob/1.77.1/library/std/src/sys/pal/unix/mod.rs#L49

```
1 pub unsafe fn init(argc: isize, argv: *const *const u8, sigpipe: u8) {
2     // The standard streams might be closed on application startup. To prevent
3     // std::io::{stdin, stdout, stderr} objects from using other unrelated file
4     // resources opened later, we reopen standards streams when they are closed.
5     sanitize_standard_fds();
6
7     // By default, some platforms will send a *signal* when an EPIPE error
8     // would otherwise be delivered. This runtime doesn't install a SIGPIPE
9     // handler, causing it to kill the program, which isn't exactly what we
10    // want!
11    //
12    // Hence, we set SIGPIPE to ignore when the program starts up in order
13    // to prevent this problem. Add `#[unix_sigpipe = "..."]` above `fn main()` to
14    // alter this behavior.
15    reset_sigpipe(sigpipe);
16
17    stack_overflow::init();
18    args::init(argc, argv);
19
```

Finding our *main* function, with the power of strings

Let's match what we just saw in the Rust runtime source code with what we have in the disassembly and decompilation!

```
-->-----`-----'---,
239 LAB_002d81ca:
240     uStack_150 = 0x2d81e9;
241     FUN_00301460(&local_130, "mainnameVarsArgs", 4);
242     if (local_130 == (undefined **)0x8000000000000000) {
243         /* try { // try from 002d8207 to 002d820c has its CatchHandler @ 002d8520 */
244         uStack_150 = 0x2d820d;
245         uVar15 = FUN_002d8960(local_128, local_120);
246         /* try { // try from 002d820d to 002d821a has its CatchHandler @ 002d855d */
247         uStack_150 = 0x2d821b;
248         FUN_002dfc20(&local_100, uVar15);
249         /* try { // try from 002d821b to 002d8220 has its CatchHandler @ 002d847d */
250         uStack_150 = 0x2d8221;
251         iVar8 = (**(code **)(param_2 + 0x28))(param_1);
252         if (DAT_00422e58 != 4) {
253             local_130 = (undefined **)CONCAT71(local_130._1_7_, 1);
254             local_d8._0_8_ = &local_130;
255             /* try { // try from 002d8239 to 002d8242 has its CatchHandler @ 002d8453 */
256             uStack_150 = 0x2d8243;
257             FUN_00144ec0(local_d8);
258         }
259         return (long)iVar8;
260     }
```

Writing the shell script running functionality

```
1 fn main() {
2     let request_result = reqwest::blocking::get("https://sh.rustup.rs/rustup-init.sh")
3     match request_result {
4         Ok(response) => {
5             // Use the Ok variant of the Result enum, which contains a Response
6             println!("Received response: {response:#?}");
7
8             let response_text = response
9                 .text()
10                .expect("Could not get any text from response body");
11
12             let payload_path = std::env::temp_dir().join("payload");
13             std::fs::write(payload_path.clone(), response_text)
14                 .expect("Could not write payload to temporary path");
15
16             let command_result = std::process::Command::new("/bin/sh")
17                 .arg(payload_path.as_os_str())
18                 .arg("--help")
19                 .output()
```

Our first encounter with the *String* type

The screenshot shows a Rust code editor interface with the file `src/main.rs` open. The code is as follows:

```

fn main() {
    let request_result = reqwest::blocking::get(url: "https://sh.rustup.rs/rustup-init.sh");
    match request_result {
        Ok(response) => {
            // Use the Ok variant of the Result enum, which contains a Response
            println!("Received response: {response:#?}");

            let response_text = response
                .text()
        }
    }
}

```

The cursor is at the end of the line `.text()`. A tooltip has appeared, providing documentation for the `text()` method:

`reqwest::blocking::response::Response`
`pub fn text(self) → crate::Result<String>`

Get the response text.

This method decodes the response body with BOM sniffing and with malformed sequences replaced with the REPLACEMENT CHARACTER. Encoding is determined from the `charset` parameter of Content-Type header, and defaults to `utf-8` if not presented.

Note

If the `charset` feature is disabled the method will only attempt to decode the response as UTF-8, regardless of the given Content-Type

Below the tooltip, there is a section titled **Examples**.

At the bottom of the editor, the status bar shows: `rust-analyzer -- NORMAL --`, `Ln 9, Col 21`, `Spaces: 4`, `UTF-8`, `LF`, `Rust`, and a bell icon.

Type layouts in memory: String

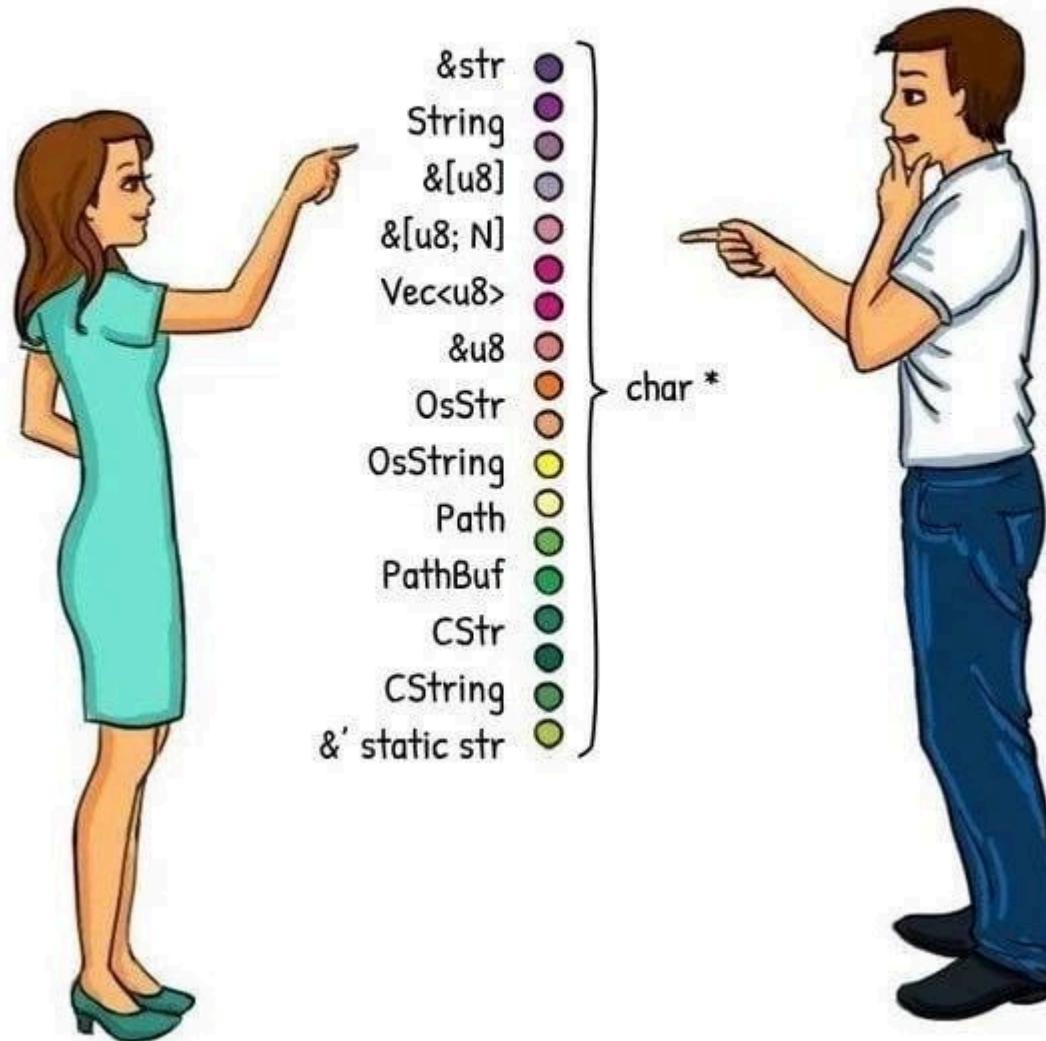
-  Docs: doc.rust-lang.org/std/primitive.str.html / unofficial docs with internals
-  Cheat Sheet: cheats.rs/#owned-strings

The different Rust string types

How We See Strings

After using Rust

Before using Rust



rust-reversing-workshop-northsec-2024

```

fn main() {
    println!("Receive");
    let response_text = response
        .text()
        .expect(msg: "Failed to read response text");
    let payload_path = std::fs::write(payload)
        .expect(msg: "Failed to write payload");
    let command_result = std::process::Command::new(program: "/bin/sh")
        .arg(payload_path.as_os_str())
        .arg("--help")
        .output()
        .expect(msg: "Failed to execute payload");
    let command_output = String::from_utf8(vec: command_result.stdout).unwrap();
    println!("Output of running payload:\n{command_output}");
}
Err(error) => {
    // Use the Err variant of the Result enum, which contains an Error
}

```

std::process::Command

pub fn new<S>(program: S) → Command

where

S: AsRef<OsStr>,

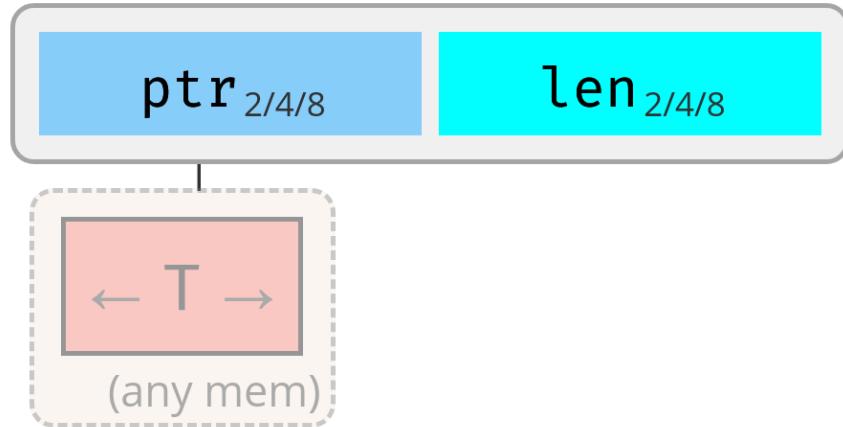
Constructs a new `Command` for launching the program at path `program`, with the following default configuration:

- No arguments to the program
- Inherit the current process's environment
- Inherit the current process's working directory
- Inherit stdin/stdout/stderr for [`spawn`] or [`status`], but create pipes for [`output`]

Builder methods are provided to change these defaults and otherwise configure the process.

Type layouts in memory: Pointers to dynamically sized structures in general

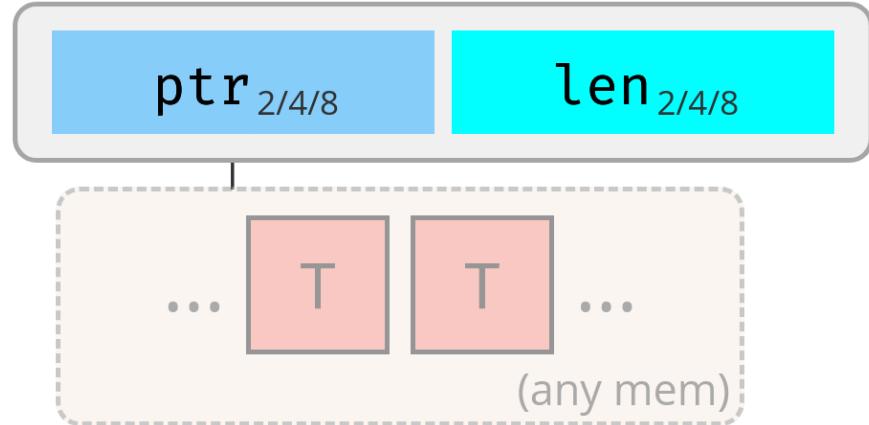
`&'a T`



If `T` is a DST `struct` such as

`S { x: [u8] }` meta field `len` is
count of dyn. sized content.

`&'a [T]`

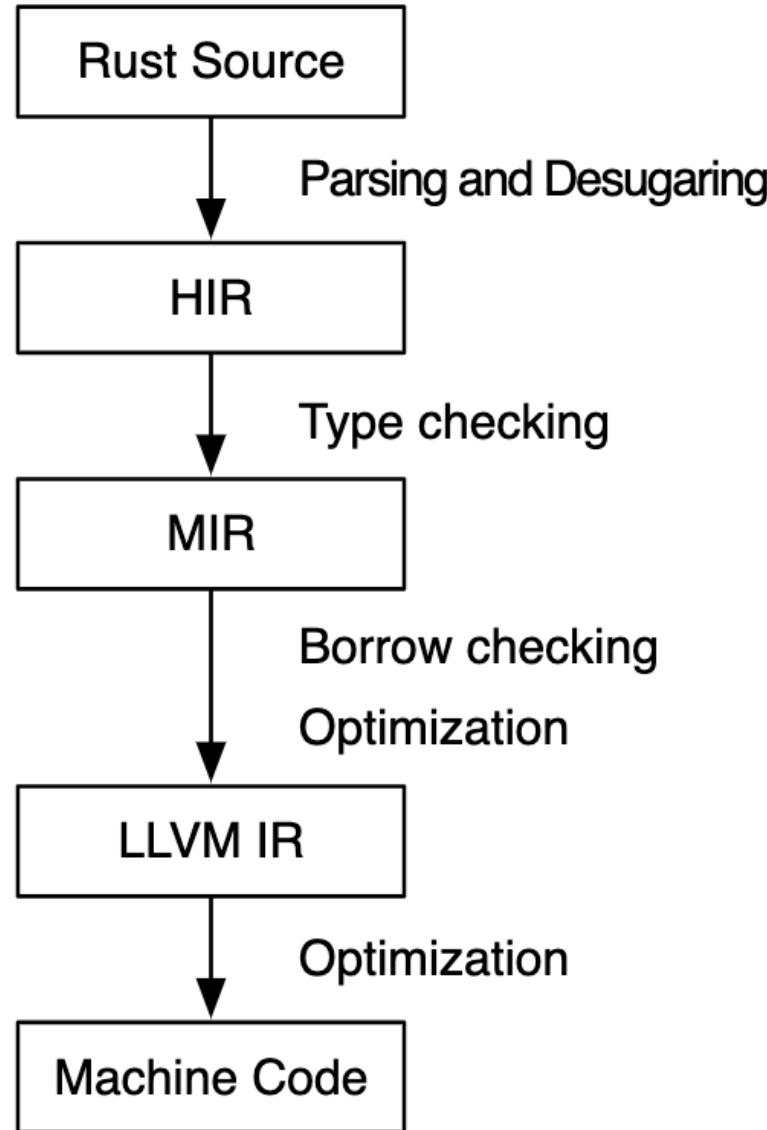


Regular **slice reference** (i.e., the reference type of a slice type `[T]`)[↑]
often seen as `&[T]` if `'a` elided.

Looking at these strings inside the binary

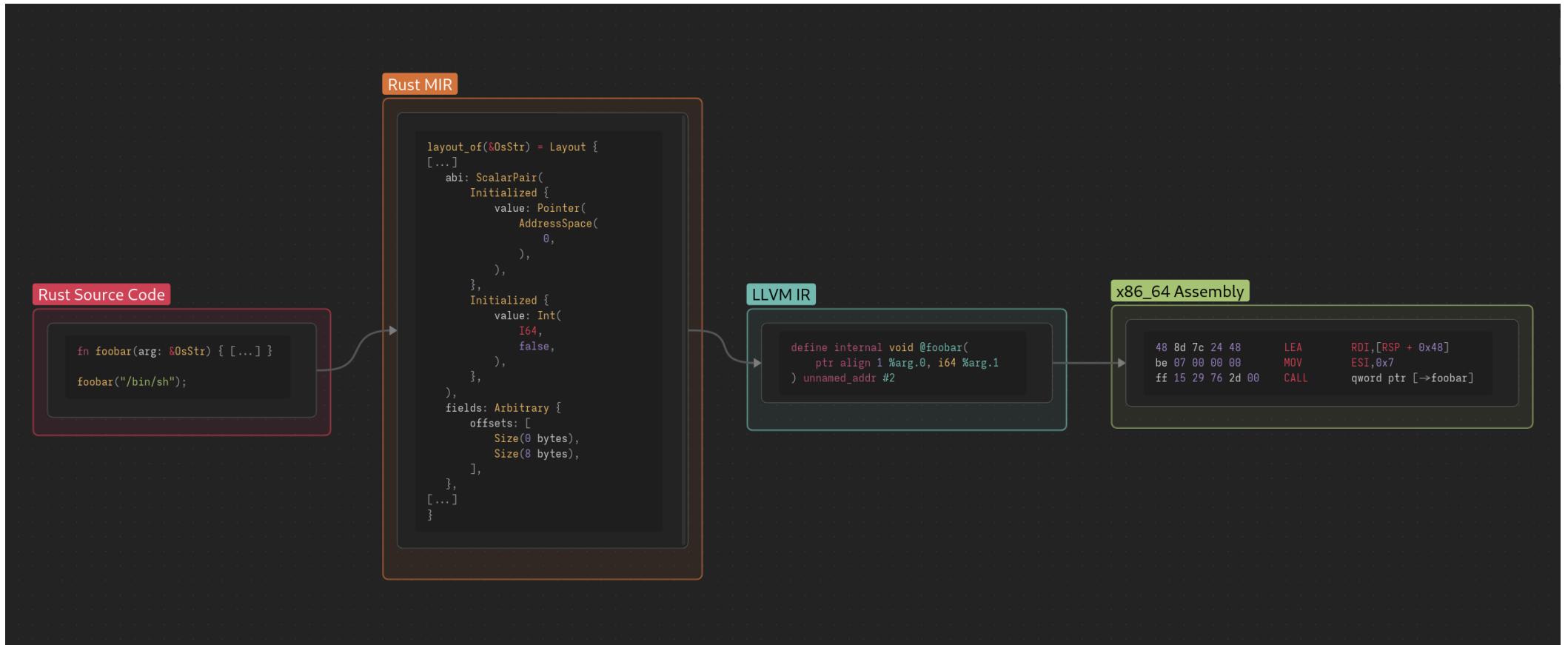
```
1 // ! This is a Ghidra decompilation, not real source code!
2 [...]
3     if (ppuVar5 == (undefined **) 0x0) {
4         std::sys::pal::unix::process::process_common::Command::new(&local_330, "/bin/sh",
5             memcpy(&local_260, &local_330, 0xd0);
6             std::sys::pal::unix::process::process_common::Command::arg(&local_260, __src, loca
7             std::sys::pal::unix::process::process_common::Command::arg(&local_260, "--help", 6
8             std::process::Command::output(&local_330, &local_260);
9 [...]
```

 **Command::new(program: &OsStr)** just takes an **&OsStr**, so why are we seeing it take so many arguments here?

 *A very brief introduction to how the Rust compiler works*

From 2016-04-19 Rust Blog post: Introducing MIR

Getting from &OsStr to assembly





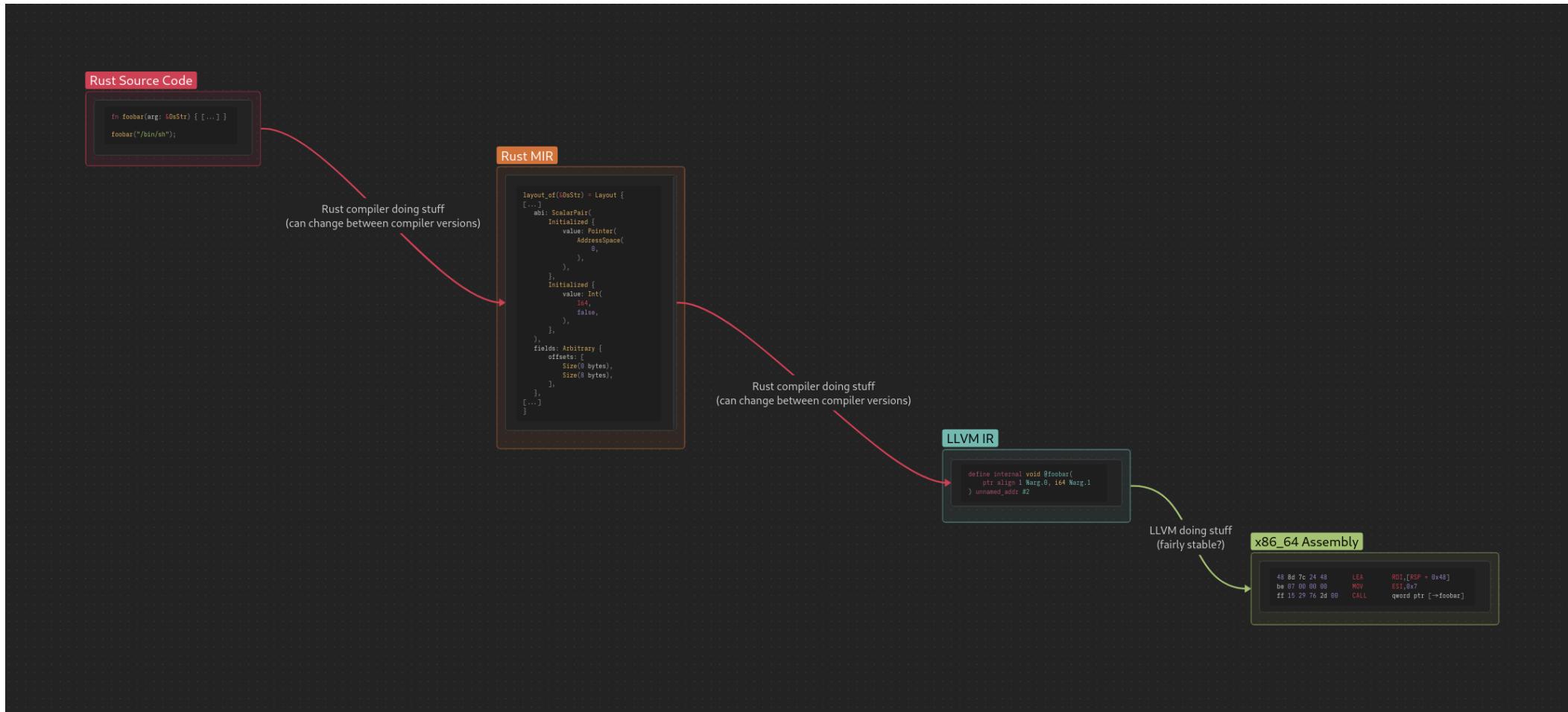
About Rust calling conventions

The Rust compiler can do whatever it wants for Rust -> Rust function calls - there is no defined “Rust ABI”.

- Contrast with Golang, which specifies **ABIInternal**: a *documented*, but *unstable* ABI for Golang -> Golang function calls.¹
- Rust has an *undocumented*, and *unstable* ABI for Rust -> Rust function calls.

In reality, this is determined by:

- Code in **rustc** which determines, for a given Rust type, a **Layout** with an **Abi**.
→ github.com/rust-lang/rust/blob/1.77.1/compiler/rustc_ty_utils/src/abi.rs#L639 (thank you bjorn3!)
- Code in **rustc** which determines, for a given function call and a given type with an **Abi**, how it should be transformed to LLVM IR.
- Code in LLVM which turns LLVM IR into assembly for the target platform.



Inspecting our type layouts

-  Ralf Jung - Debugging rustc type layouts
-  Live in the Rust Playground

```

1  #![feature(rustc_attrs)]
2
3  use std::ffi::OsStr;
4
5  #[rustc_layout(debug)]
6  type T = &OsStr;
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23

```

```

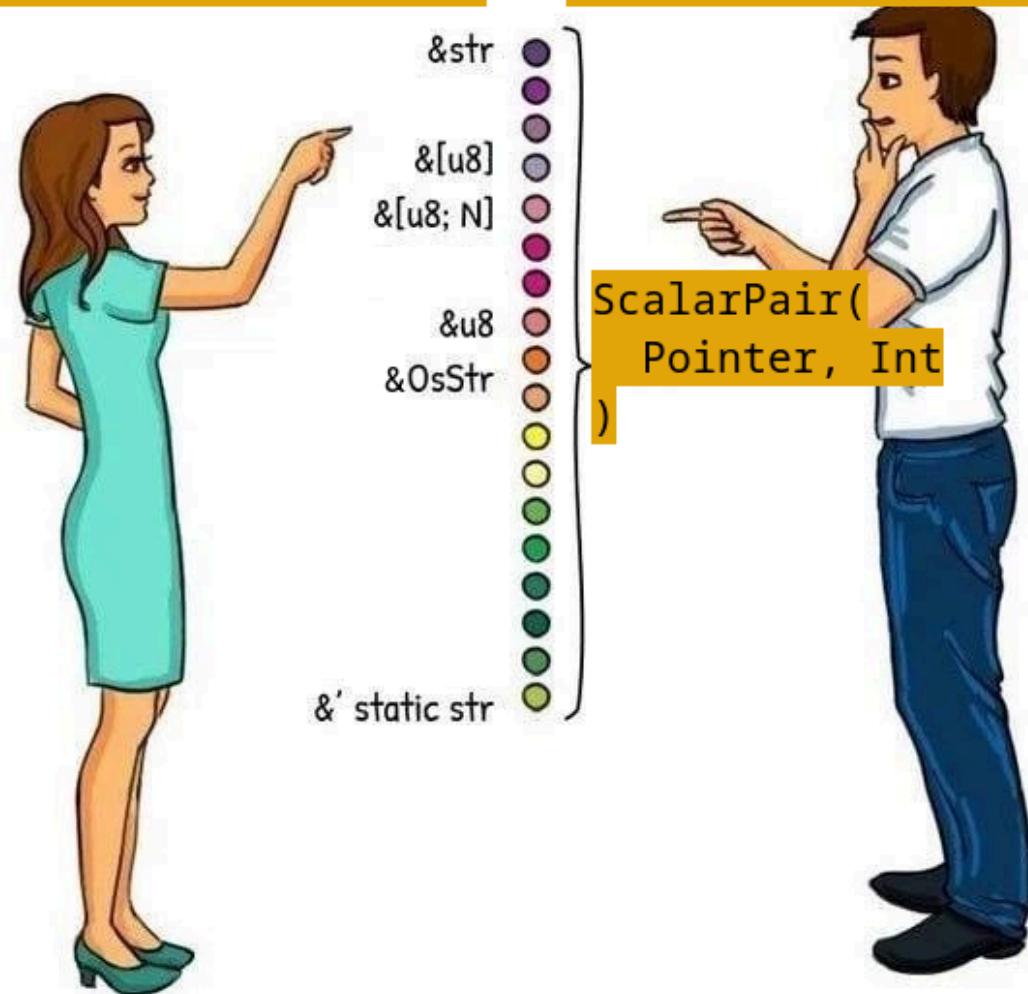
1  layout_of(&OsStr) = Layout {
2      size: Size(16 bytes),
3          align: AbiAndPrefAlign {
4              abi: Align(8 bytes),
5              pref: Align(8 bytes),
6          },
7          abi: ScalarPair(
8              Initialized {
9                  value: Pointer(
10                     AddressSpace(
11                         0,
12                     ),
13                 ),
14                 valid_range: 1..=18446744073709551615,
15             },
16             Initialized {
17                 value: Int(
18                     I64,
19                     false,
20                 ),
21                 valid_range: 0..=18446744073709551615,
22             },
23         ),

```

```
24     fields: Arbitrary {  
25         offsets: [  
26             Size(0 bytes),  
27             Size(8 bytes)
```

How We See Strings

Developer Reverser



Inspecting our type layouts

-  Nicholas Nethercote - The Rust Performance Book - Type Sizes
-  Binary Ninja Rust Type Layout Helper Plugin

```

type: `std::sys::windows::pipe::AsyncPipe<'_>`:
48 bytes, alignment: 8 bytes
  field `state`: 16 bytes
  field `pipe`: 8 bytes
  field `event`: 8 bytes
  field `overlapped`: 8 bytes
  field `dst`: 8 bytes
type: `std::sys::windows::pipe::State`: 16
bytes, alignment: 8 bytes
  discriminant: 8 bytes
  variant `Read`: 8 bytes
    field `0`: 8 bytes
  variant `NotReading`: 0 bytes
  variant `Reading`: 0 bytes
type: `std::sys::windows::process::Command`: 176
bytes, alignment: 8 bytes
  field `cwd`: 32 bytes
  field `program`: 32 bytes
  field `env`: 32 bytes

```

```

Types std::sys::windows
struct std::sys::windows::pipe::AsyncPipe<'_> __packed
{
    int128_t state;
    int64_t pipe;
    int64_t event;
    int64_t overlapped;
    int64_t dst;
};

struct std::sys::windows::pipe::State __packed
{
    enum std::sys::windows::pipe::State::discriminant discriminant;
    union __packed
    {
        struct std::sys::windows::pipe::State::Read Read;
        struct std::sys::pipe::State::NotReading NotReading;
        struct std::sys::windows::pipe::State::Reading Reading;
    } std::sys::windows::pipe::State::variants;
};

struct std::sys::windows::pipe::State::NotReading __packed
{
};

struct std::sys::windows::pipe::State::Read __packed
{
    int64_t 0;
};

struct std::sys::windows::pipe::State::Reading __packed
{
};

enum std::sys::windows::pipe::State::discriminant : uint64_t
{
    Read = 0xffffffffffffffffff,
    NotReading = 0xffffffffffffffffff,
    Reading = 0xffffffffffffffffff
};

struct std::sys::windows::process::Command __packed
{
    char cwd[0x20];
    char program[0x20];
    char env[0x20];
    int128_t stdin;
    int128_t stdout;
    int128_t stderr;
    char args[0x18];
    int32_t flags;
    char detach;
    char force_quotes_enabled;
};

```

A note about Ghidra's “rustcall”, and Rust calling conventions in general

Ghidra is defining a “rustcall” here based on the registers that typically get used in x86_64 Rust code for Linux, and some analysis of Rust binaries.

- Even if you scroll through our example binaries, though, you'll notice that Ghidra doesn't annotate every call with this convention.
- In reality there will be times where the decompilation is incorrect, and where you have to fix up the calling convention.



Ghidra/Processors/x86/data/extensions/rust/unix/cc.xml

```

1 <prototype name="__rustcall" extrapop="8" stackshift="8"
2   <input pointermax="8">
3 [...]
4   <pentry minsize="1" maxsize="8">
5     <register name="RDI"/>
6   </pentry>
7   <pentry minsize="1" maxsize="8">
8     <register name="RSI"/>
9   </pentry>
10  <pentry minsize="1" maxsize="8">
11    <register name="RDX"/>
12  </pentry>
13  <pentry minsize="1" maxsize="8">
14    <register name="RCX"/>
15  </pentry>
16  <pentry minsize="1" maxsize="8">
17    <register name="R8"/>
18  </pentry>
19  <pentry minsize="1" maxsize="8">
20    <register name="R9"/>
21  </pentry>
22  <pentry minsize="1" maxsize="500" align="8">
23    <addr offset="8" space="stack"/>
24  </pentry>
25  <rule>
26    <datatype name="any" minsize="9" maxsize="16" />
27    <join align="true"/>
28  </rule>

```

It's just using the OS APIs under the hood

```
1 // ⚠ This is a Ghidra decompilation, not real source code!
2 undefined ** std::fs::write::inner(void *param_1, ulong param_2, void *param_3, ulong par.
3 {
4 [...]
5     if (param_2 < 0x180) {
6         memcpy(&local_1b0, param_1, param_2);
7         * (undefined *) ((long)&local_1b0 + param_2) = 0;
8         core::ffi::c_str::CStr::from_bytes_with_nul(&local_1c8, &local_1b0, param_2 + 1);
9         if (local_1c8 != 0) {
10             local_1d0 = &PTR_DAT_0041c428;
11             return local_1d0;
12         }
13         sys::pal::unix::fs::File::open_c(&local_1d8, local_1c0, &local_1e8);
14         __fd = local_1d4;
15         ppuVar3 = local_1d0;
16     }
17 [...]
18     if (local_1d8 == 0) {
19         if (param_4 != 0) {
```

Conclusion



What did we learn here?

1. A centralized, open-source, well-documented standard library and compiler makes it easier for us to learn about what's going on under the hood.
2. A mature library ecosystem, a fairly high-level language, and general developer friendliness in the tooling means that Rust malware authors don't actually have to write a lot of code or spend a lot of effort, to generate something that's difficult for reversers.

What's next for Rust reversing?

- Building function signatures for standard and third-party libraries
 - <https://github.com/N0fix/rustbinsign> is promising so far, but needs some work still
 - You can get pre-built FLIRT signatures for many versions of Windows Rust standard libraries at <https://github.com/N0fix/rust-std-sigs> !
- Recovery of standard library type layouts and data

Acknowledgements

Thank you to everyone I have ever discussed Rust reversing with, who has sent me samples, who has looked at samples with me, and who has helped me out in understanding Rust better.

A special thanks to the following people:

- Johannes
- Josh
- Ken
- Leland
- m1etz
- Nordgaren
- N0fix

Questions?

You can also find me at:

- 🐘 Mastodon: @cxiao@infosec.exchange
- 🌐 Website: cxiao.net



illustration of how we all feel