



Securing the Rust supply-chain ecosystem

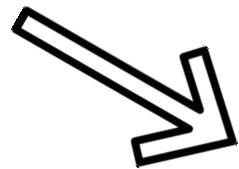
Caleb Stanford

With: David Thien, Lydia Zoghbi, Ranjit Jhala, Deian Stefan,
and others

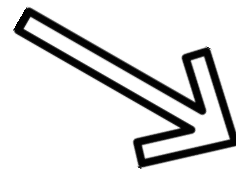
UC San Diego

About Me

PL for Systems



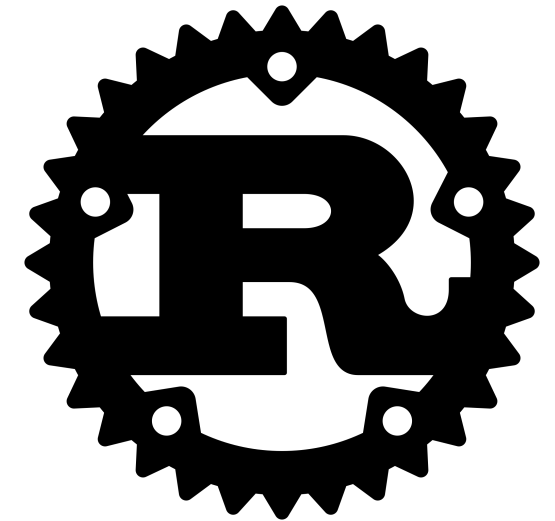
UC San Diego



UC DAVIS
UNIVERSITY OF CALIFORNIA

Starting in July!

Rust: no longer Mozilla's esoteric project



19

28



Rust

0.63%

+0.18%

Linus Torvalds: Rust will go into Linux 6.1

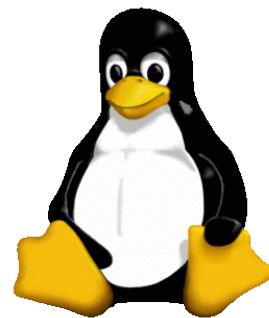
At the Kernel Maintainers Summit, the question wasn't, "Would Rust make it into Linux?" Instead, it was, "What to do about its compilers?"



Supporting the Use of Rust in the Chromium Project

January 12, 2023

Posted by Dana Jansens (she/her), Chrome Security Team



Rust in the Android platform

April 6, 2021



Modernizing embedded
development on Azure Sphere
with Rust

[Developer Center / ...](#)

Rust on AWS

Tools, docs, and sample code to develop
applications on the AWS cloud

Why Rust?

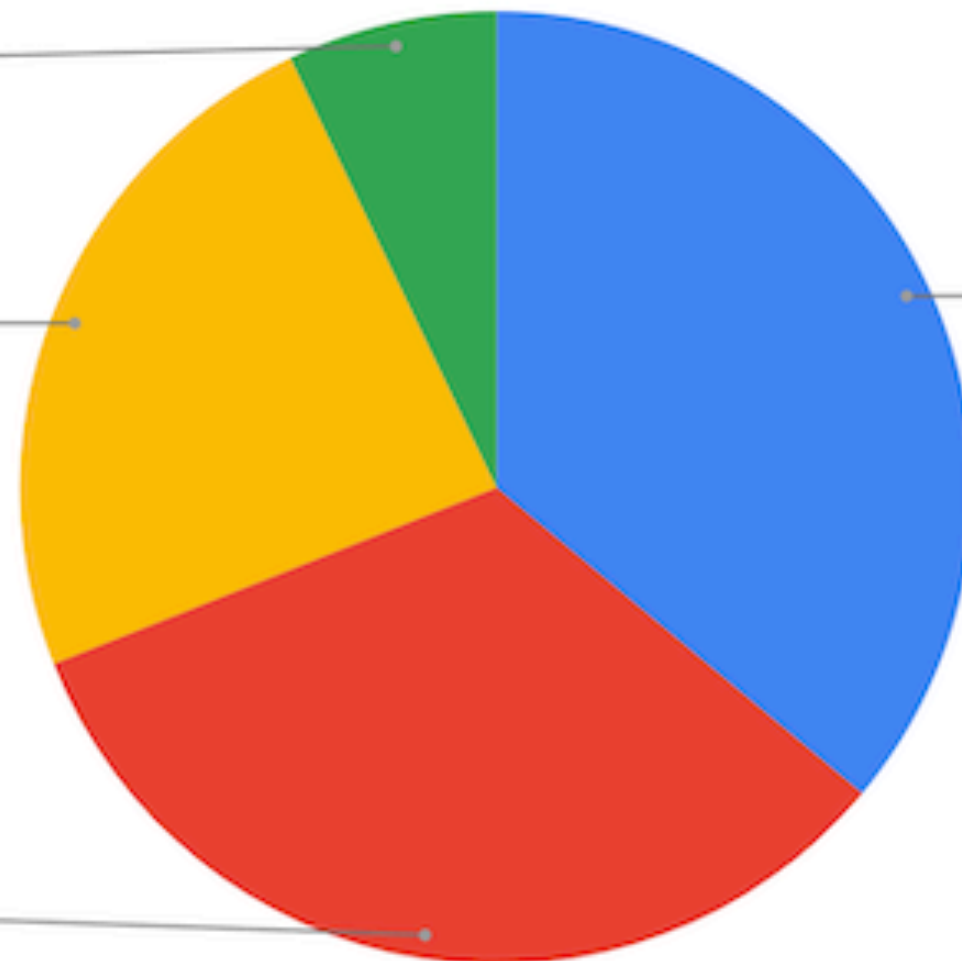
Memory Safety

High+, impacting stable

Security-related assert
7.1%

Other
23.9%

Other memory unsafety
32.9%



Use-after-free
36.1%



Future of Memory Safety

Challenges and Recommendations

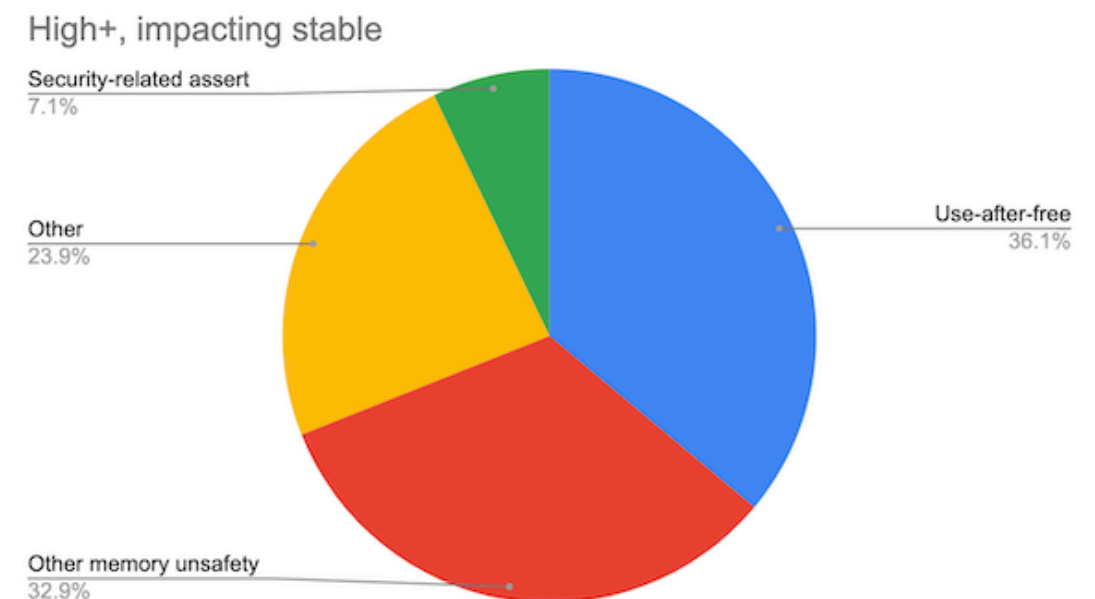
Yael Grauer
January 2023



Security Planner

Why Rust?

- Rust is *memory and type safe*



- Amazing tooling: testing, docs, etc.

Why Rust?

- Rust is *memory and type safe*



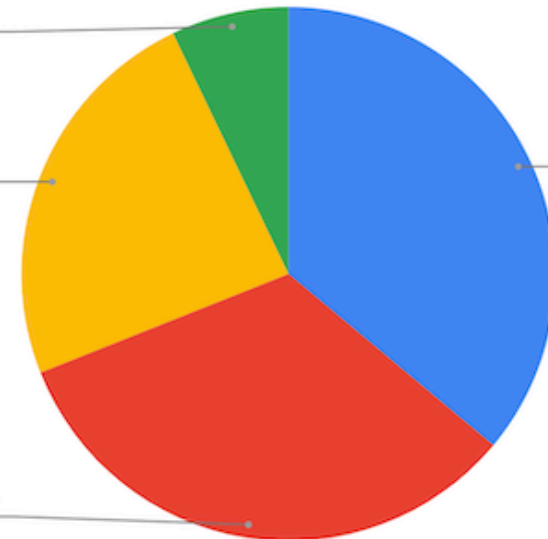
High+, impacting stable

Security-related assert
7.1%

Other
23.9%

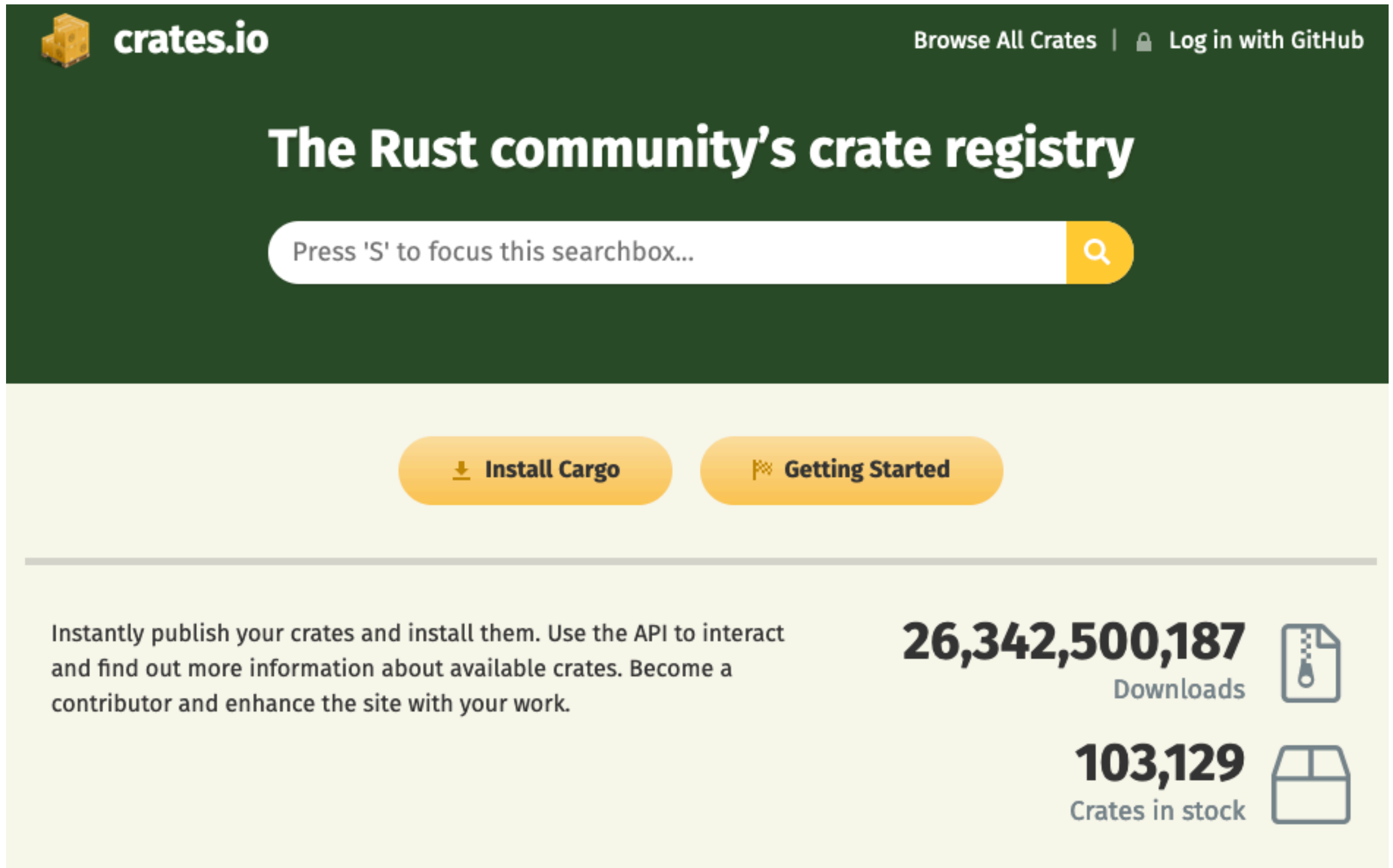
Other memory unsafety
32.9%

Use-after-free
36.1%



- Amazing tooling: testing, docs, etc.

Third-party code!



The screenshot shows the crates.io website. At the top left is the crates.io logo. At the top right are links for "Browse All Crates" and "Log in with GitHub". The main heading is "The Rust community's crate registry". Below this is a search bar with the placeholder text "Press 'S' to focus this searchbox..." and a magnifying glass icon. Under the search bar are two buttons: "Install Cargo" and "Getting Started". A horizontal line separates this section from the bottom. The bottom section contains a paragraph of text on the left and two statistics on the right. The first statistic is "26,342,500,187 Downloads" with a document icon. The second statistic is "103,129 Crates in stock" with a crate icon.

crates.io Browse All Crates | Log in with GitHub

The Rust community's crate registry

Press 'S' to focus this searchbox...

[Install Cargo](#) [Getting Started](#)

Instantly publish your crates and install them. Use the API to interact and find out more information about available crates. Become a contributor and enhance the site with your work.

26,342,500,187
Downloads

103,129
Crates in stock

Problem: third-party code

- Rust is not actually memory safe!
 - Real code uses unsafe, calls C foreign functions, etc.
- Memory safety != isolation
 - Third party crate can read/write to fs, net, etc.
 - At run time, at build time

Problem: third-party code

Security advisory: malicious crate `rustdecimal`

May 10, 2022 · The Rust Security Response WG

The crate contained identical source code and functionality as the legit `rust_decimal` crate, except for the `Decimal::new` function.

When the function was called, it checked whether the `GITLAB_CI` environment variable was set, and if so it downloaded a binary payload into `/tmp/git-updater.bin` and executed it. The binary payload supported both Linux and macOS, but not Windows.

Audit your supply chain!

Cargo Vet

The `cargo vet` subcommand is a tool to help projects ensure that third-party Rust dependencies have been audited by a trusted entity. It strives to be lightweight and easy to integrate.





Showing 2,172 changed files with 12,009 additions and 11,895 deletions.



Auditing Sucks.

PL to the rescue!

1. Audit the code that matters.
2. Make audits re-usable and upgradable.
3. Enforce when you can't/don't want to audit.

1. Audit the code that matters

Some data

Top 1000 crates on crates.io

```
./scan.py -i data/crate-lists/top10000.csv -o top10000
```

```
==== Crate Totals ====  
3046 crates with 1 or more dangerous imports  
6952 crates with 0 dangerous imports
```

Risky function calls by crate

```
==== Crate Summary ====  
Number of dangerous imports by crate:  
sysinfo: 300  
nix: 223  
rusoto_credential: 129  
trust-dns-proto: 123  
tokio: 113  
git2: 96  
vcpkg: 92  
crossterm: 92  
cc: 73  
rocksdb: 71  
ipnet: 67  
filetime: 60  
ipnetwork: 56
```

Some data

Top 1000 crates on crates.io

```
./scan.py -i data/crate-lists/top10000.csv -o top10000
```

```
==== Crate Totals ====  
3046 crates with 1 or more dangerous imports  
6952 crates with 0 dangerous imports
```

Most crates do not have
any risky effects!

Risky function calls by crate

```
==== Crate Summary ====  
Number of dangerous imports by crate:  
sysinfo: 300  
nix: 223  
rusoto_credential: 129  
trust-dns-proto: 123  
tokio: 113  
git2: 96  
vcpkg: 92  
crossterm: 92  
cc: 73  
rocksdb: 71  
ipnet: 67  
filetime: 60  
ipnetwork: 56
```

Among crates with risk, most
are concentrated in a small
number of “core” crates

Mozilla data



- Crates marked “exempt” in Cargo Vet

```
==== Crate Totals ====  
82 crates with 1 or more dangerous imports  
235 crates with 0 dangerous imports
```

Static analysis

- Only look at paths that lead to “powerful” APIs
- Only audit subset of a library you’re actually using

Static analysis

- Only look at paths that lead to “powerful” APIs

Side effects:

Behavior not visible in the type



Safe Haskell

Small anecdote

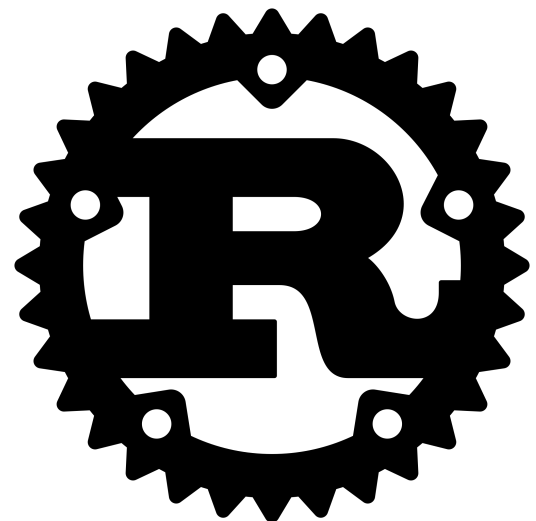
Memory safety, *no wild pointers*.

Typestate system, *no null pointers*.

Mutability control, *immutable by default*.

Side-effect control, *pure by default*.

— Graydon Hoare, 2010



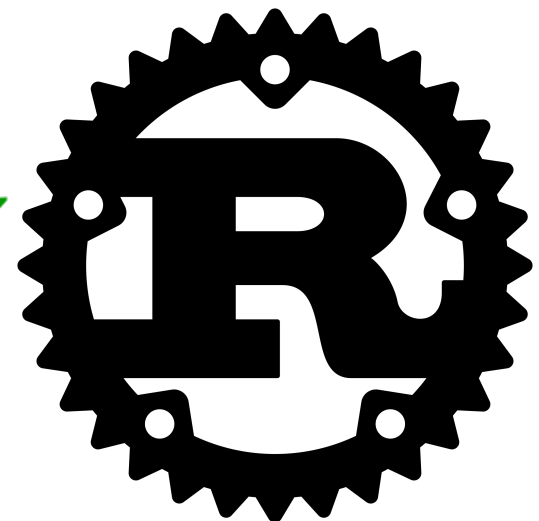
Small anecdote

Memory safety, *no wild pointers*. ✓

Typestate system, *no null pointers*. ✓

Mutability control, *immutable by default*. ✓

Side-effect control, *pure by default*. ✗



— Graydon Hoare, 2010

Static analysis

- Only look at paths that lead to “powerful” APIs

Taint analysis:

Sources = *Side effects* + memory unsafety

Static analysis

- Only audit subset of a library you're actually using

Taint analysis:

Sinks = `main()` and publicly exposed library functions

The Dependency Problem

- A uses B uses C uses D; D accesses filesystem
- “Depth” of analysis.
 - Display effects in D to C only?
 - Display effects in D to A, B, and C?

Modular Auditing: Delegation

```
? Select how to mark this effect:  
(s)afe, (u)nsafe, (c)aller checked, ask me (l)ater, e(x)it tool  
█
```

- Not enough information to assess safety for ALL possible callers
- **Delegation:** effects as *caller- or callee-checked*
- (Done routinely by auditors all the time)
- Solves the dependency problem!

Demo

Cargo Scan

`cargo scan` is an under-development tool to scan and analyze Cargo (Rust) dependencies for security and auditing purposes. It can also be used in tandem with [cargo vet](#).



- Static-analysis assisted supply chain auditing tool
- Built in Rust

Demo

Cargo Scan

`cargo scan` is an under-development tool to scan and analyze Cargo (Rust) dependencies for security and auditing purposes. It can also be used in tandem with [cargo vet](#).

Demo

```
[caleb@caleb-mac rust-src % cargo run --bin check_package ../data/packages/rustc_version ../rustc_version.policy
Finished dev [unoptimized + debuginfo] target(s) in 0.06s
Running `target/debug/check_package ../data/packages/rustc_version ../rustc_version.policy`
Creating new policy file

=====

error: effect: Some(Sink(Pattern(Path("std::env"))))
lib.rs:201:1

197
198   /// Returns the `rustc` SemVer version and additional metadata
199   /// like the git short hash and build date.
200   pub fn version_meta() -> Result<VersionMeta> {
201       let cmd = env::var_os("RUSTC").unwrap_or_else(|| OsString::from("rustc"));
202   }

? Select how to mark this effect:
(s)afe, (u)nsafe, (c)aller checked, ask me (l)ater, e(x)it tool

```

Function which has a possibly
unsafe side effect

Options for auditor

Side effect

2. Make audits re-usable and upgradable

 Showing 2,172 changed files with 12,009 additions and 11,895 deletions.



2. Make audits re-usable and upgradable

Showing 2,172 changed files with 12,009 additions and 11,895 deletions.



Audits can be exhausting

The “LGTM” effect

Showing 2,172 changed files with 12,009 additions and 11,895 deletions.

Diff settings ▾ Review changes ▾

Submit your review

Review summary

LGTM

LGTM 👍

☒ **Comment**
Submit general feedback without explicit approval.

☐ **Approve**
Submit feedback and approve merging these changes.

☐ **Request changes**
Submit feedback that must be addressed before merging.

Submit review



Audits can be exhausting

Audit exhaustion: code changes

- Code changes => rerun the taint analysis for ALL source functions
 - And across all client crates transitively dependent on the code!

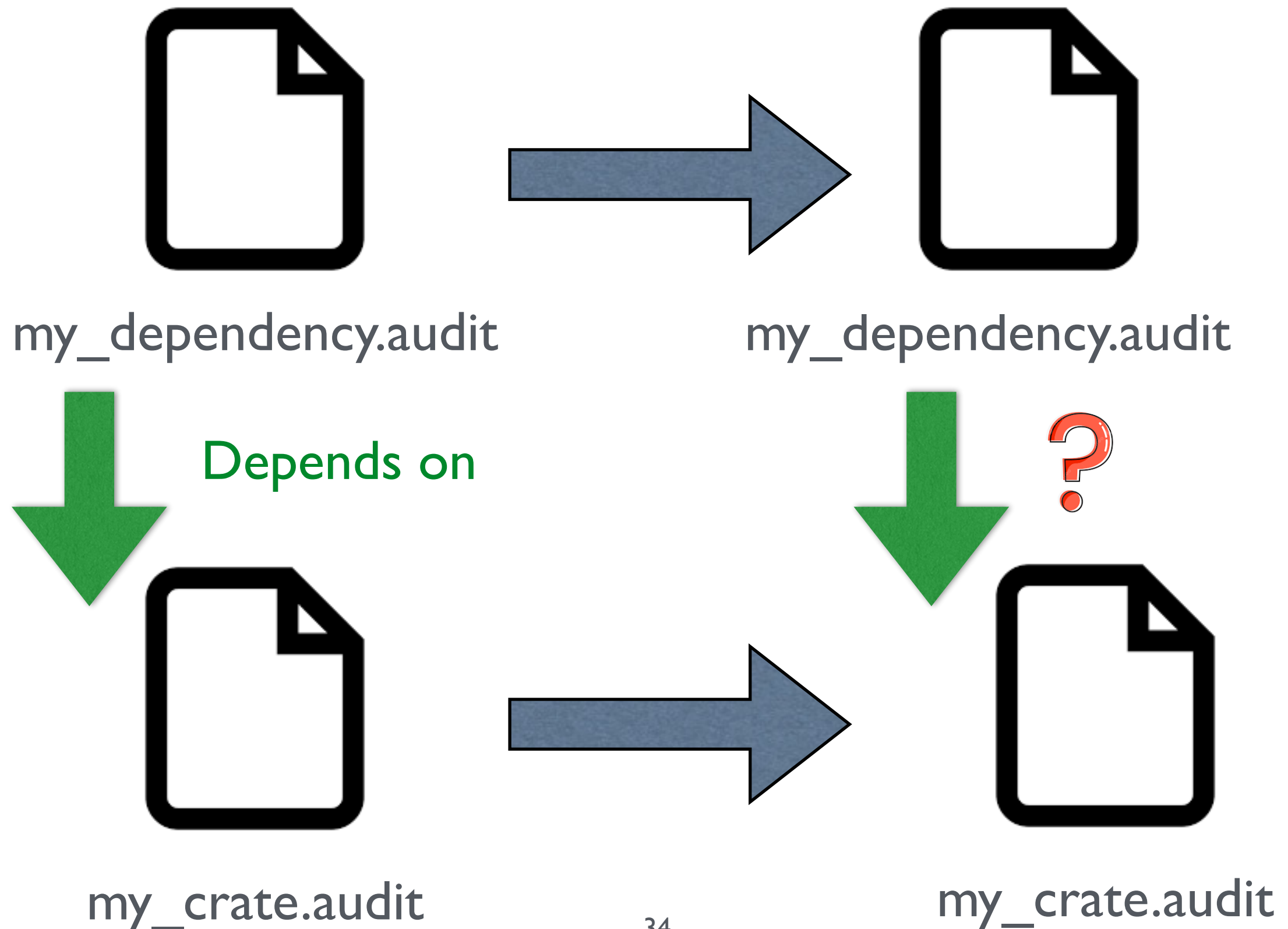
Solution

- *Save partial results* to continue an audit later
- *Update audit* on diff

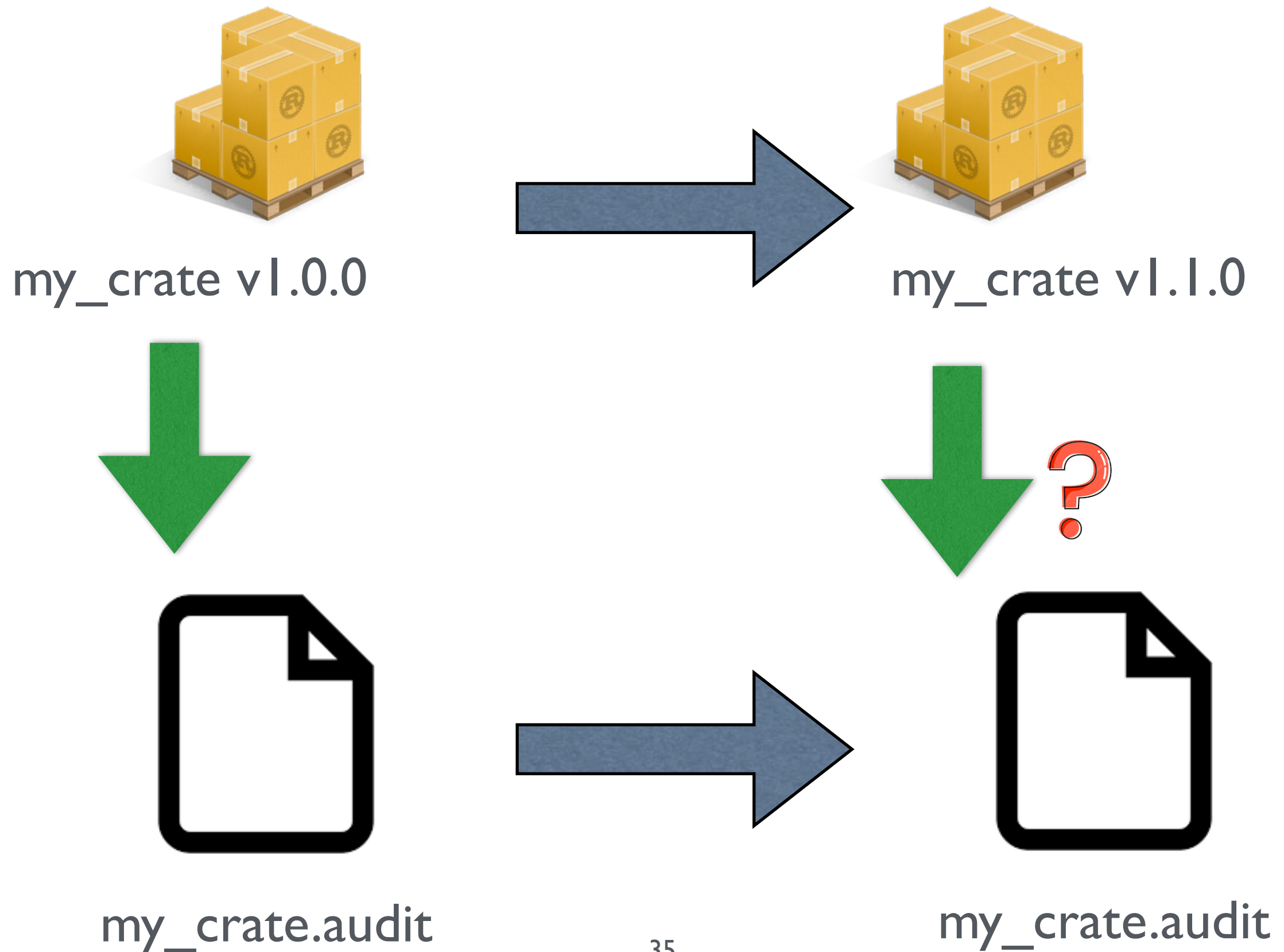


my_crate.audit

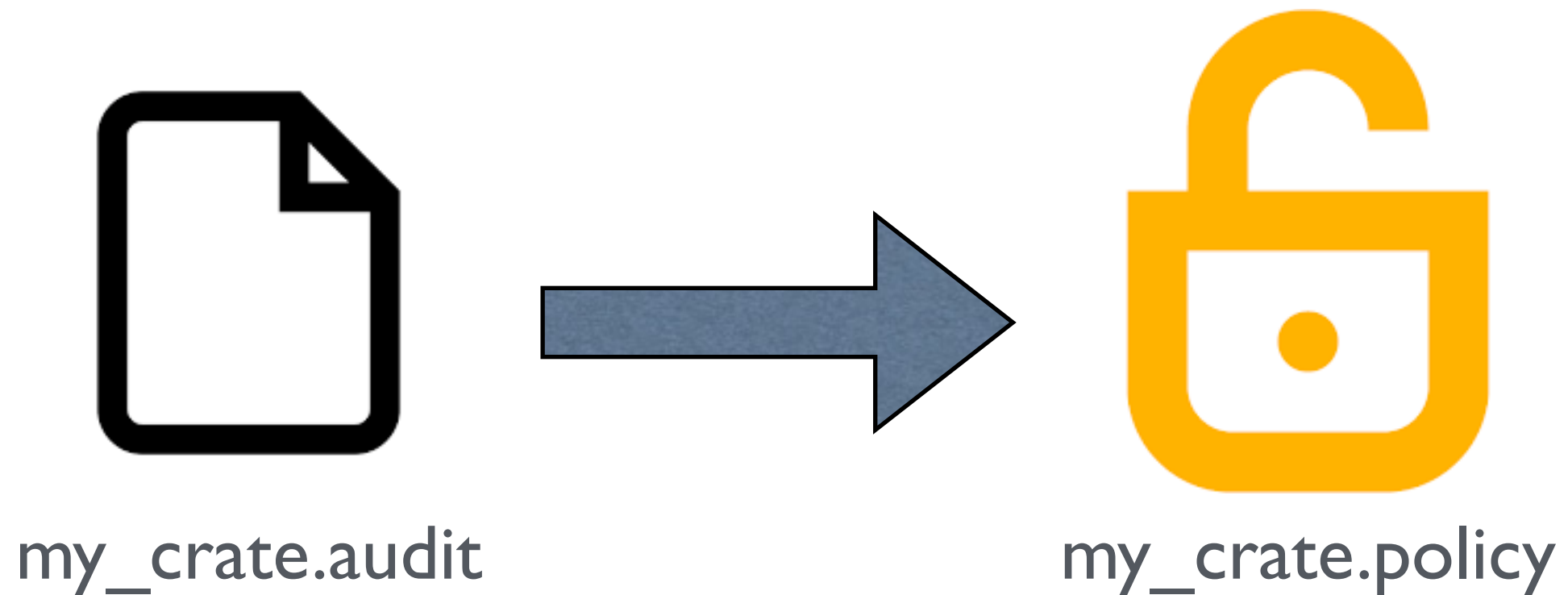
The audit update problem (v1)



The audit update problem (v2)



3. Enforce when you can't/don't want to audit

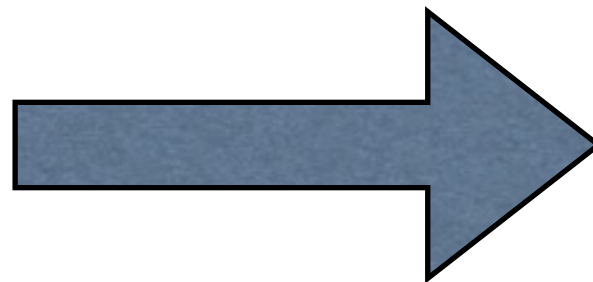


- Use policy specification to run library with least privs

3. Enforce when you can't/don't want to audit



my_crate.audit



my_crate.policy



RLBox

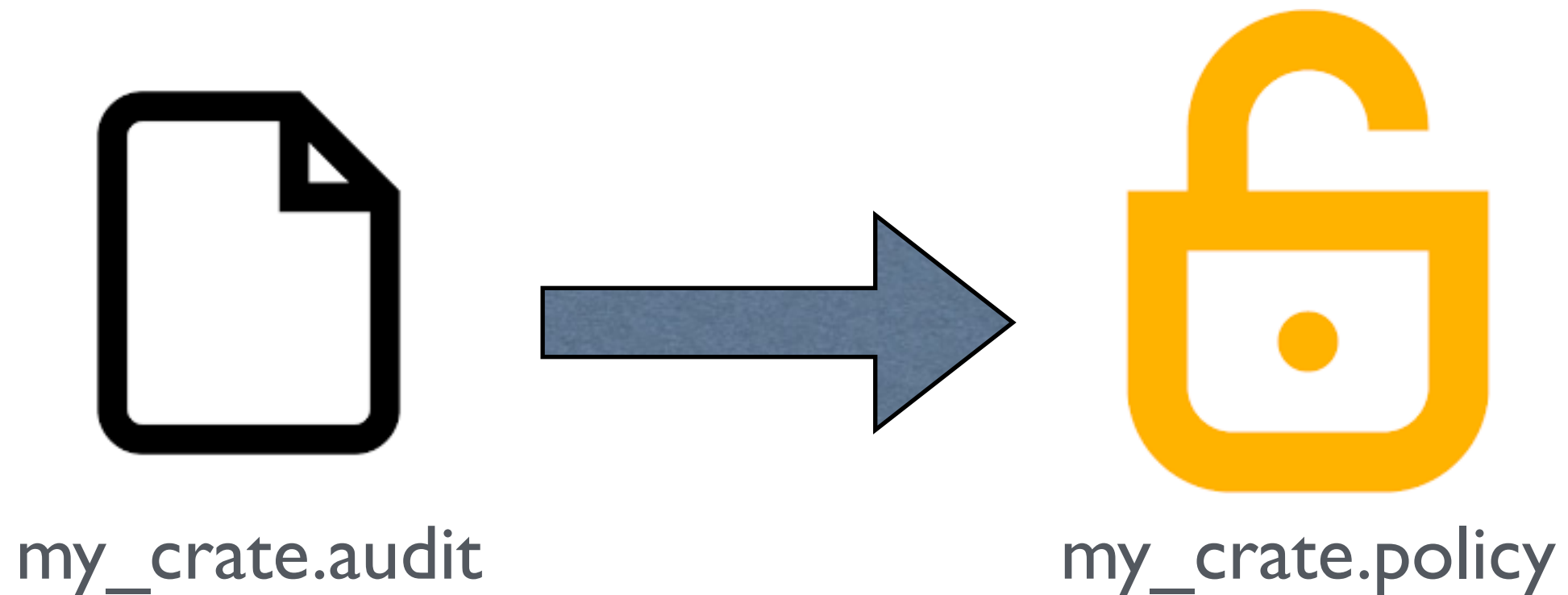
Lightweight sandboxing for C FFI code

Why? Difficult edge cases

- Macro expansion
- Build-time effects (build.rs)
- Dynamic dispatch (trait objects)
- Closures and function pointers

(Current solution: report to the user)

3. Enforce when you can't/don't want to audit



Information from static analysis
- don't need to sandbox everything!

Other interesting research challenges

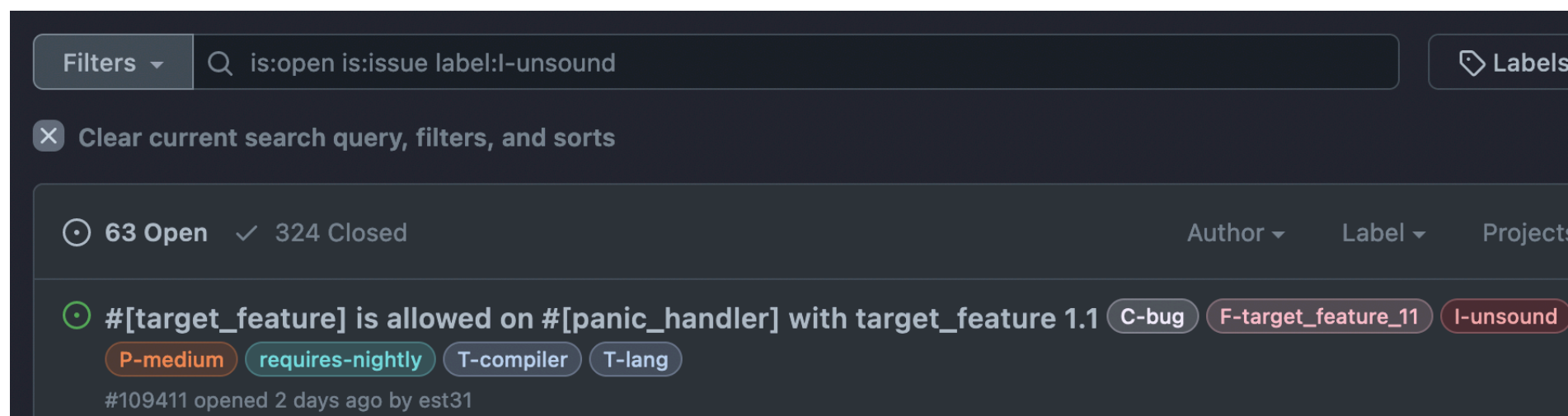
- Build-time effects

```
use my_macros::insert_backdoor;

#[derive(insert_backdoor)]
struct Password {
    pass: String
}
```

- Higher-order functions

- Unsoundness in the Rust compiler



Summary

1. Audit the code that matters
2. Make audits re-usable and upgradable
3. Enforce when you can't/don't want to audit

- Try out Cargo Scan!

github.com/PLSysSec/cargo-scan

