Rust: it's memory-safe, so what can still go wrong?

Before your next release, forbid `unsafe` where you don't need it, run the rest under Miri, turn on release overflow checks and keep untrusted input away from panics. Judge everything else by whether it guards what the compiler can't check: `unsafe` code, FFI, sizes and paths from input, and code that runs at build time.

Reviewed	Next review	Version	Written for
3 Nov 2026	**3 Dec 2026**	**1.0**	**Developers and tech leads writing production Rust, and the security people who review it**

Latest version
sheets.einblick.dev/secure-rust/

If you only do five things

Seven of the 2025 CWE Top 25 are buffer overflows, out-of-bounds access, use after free or NULL dereference[1]. Android estimates 0.2 memory-safety vulnerabilities per million lines of its Rust, based on one near-miss in about 5 million lines, against about 1,000 in its C and C++[2,3]. Panics, overflow, deadlocks and logic errors remain[3].

1. **Forbid unsafe code wherever you can.**
   Set `unsafe_code = "forbid"` under `[lints.rust]`[4] in crates that don't need it, and keep the rest in small modules behind a safe API.
2. **Justify and test every unsafe block.**
   Require `// SAFETY:` comments with Clippy's `undocumented_unsafe_blocks`[5] and run the tests under `cargo +nightly miri test`[6].
3. **Turn on overflow checks in release builds.**
   Release builds don't check overflow by default[8]; in `bytes`, a wrapped `new_cap + offset` could create out-of-bounds slices[9]. Set `overflow-checks = true` in `[profile.release]`.
4. **Keep attacker input away from panics.**
   `unwrap()`, indexing, slicing and division can panic[5], unwinding the thread or, under `panic = "abort"`, ending the process[10]. `bcrypt::verify` panicked on hashes with multi-byte characters[11]. On input paths use `get()`, `?` and `checked_*`.
5. **Treat a build as running dependency code.**
   Build scripts and proc macros run with the compiler's access to your files[12]; a compromised `arrayref` release pulled in a build script that fetched malware[13]. Keep secrets out of build jobs and gate CI on `cargo deny check` for known-bad crates[14].

Review a Rust pull request

1. **New unsafe code, extern block or unsafe impl?**
   Check each SAFETY comment against every caller and thread, check that a panic midway can't leave a broken state, and ask for a Miri test unless the path crosses FFI.
2. **Can outside input reach a panic?**
   Follow it to `unwrap`, `expect`, `[]`, slices and `/`. Ask for `get`, `?` or a checked method on that path.
3. **Is every size computed with checked arithmetic?**
   Look for `+`, `*` and `as` casts on lengths, offsets and capacities, and for lengths read from the input that reach an allocation.
4. **Does input reach a process, a path or a query?**
   Check `Command`, `Path::join` and archive extraction, then SQL, HTML and authorization, where Rust doesn't help.
5. **New or updated dependency?**
   Read the published files in crates.io's Code tab, which can differ from the repository[15]. Start with `build.rs` and procedural macros.

Rust traps and what to use instead

- **`unsafe` that overlooks other threads.** Rust Binder's SAFETY comment assumed an entry was in its owner's list or in none, but `Node::release` had moved it, so two threads corrupted the same pointers[7]. Check each claim against every caller and thread, one operation per block (`multiple_unsafe_ops_per_block`)[5].
- **Unsafe cleanup that a panic interrupts.** `thin-vec` reset its length only after dropping the elements, so a panicking `Drop` let safe code cause a double free. Reset the length first or use a drop guard, as `std` does[16].
- **Lengths read from the input.** A 512-byte file made a parser call `Vec::with_capacity` for 9.3 GB; under a memory limit the process aborts, and no `Result` catches that. Capping the input size didn't help: check each length against the bytes left[17].
- **`Command` running a `.bat` or `.cmd` file on Windows.** `cmd.exe` splits arguments its own way, so untrusted ones could run commands before Rust 1.77.2 (CVE-2024-24576, critical), or 1.81.0 if the name ended in a dot or space[19]. Handle the `InvalidInput` error `spawn` now returns, and never pass untrusted input to `raw_arg`[18].
- **`base.join(user_path)`.** An absolute `user_path` replaces the base[20]; accept only `Component::Normal` parts. In archives, `tar` before 0.4.45 followed a planted symlink and changed permissions outside the extraction root[21].
- **FFI boundaries.** Rust 2024 requires `unsafe extern` because a wrong signature in an `extern` block can cause undefined behavior[22], as can a C++ exception unwinding into Rust through `extern "C"`[10]. Use `"C-unwind"` there, and `catch_unwind` in functions you export.
- **Opening a repository you haven't reviewed.** Cargo can execute code from the project it runs in[23], and rust-analyzer runs build scripts and procedural macros by default[24]. Open strangers' pull requests in a VM without credentials.

Tools that catch it

- **rustc lints.** Set `unsafe_code = "forbid"` and `unsafe_op_in_unsafe_fn = "deny"` under `[lints.rust]`[4].
- **Clippy restriction lints,** off by default: `undocumented_unsafe_blocks`, `multiple_unsafe_ops_per_block`, `unwrap_used`, `indexing_slicing`, `string_slice` and `arithmetic_side_effects`[5].
- **Miri** finds out-of-bounds access, use after free, data races and aliasing violations, but only in the executions your tests reach, and not across FFI[6].
- **`cargo fuzz`** for parsers. **Kani** checks small `unsafe` cores for panics, overflow and memory errors on all harness inputs, not concurrency[25].
- **`cargo audit`** checks `Cargo.lock` and `cargo auditable` binaries against RustSec[23]. **`cargo deny`** adds bans, licenses and allowed sources[14]. **`cargo vet`** checks dependencies against audits you trust[26].

What you can skip

- **Banning `unsafe` outright.** FFI and hardware need it. Google calls a ban unworkable and is adding training on writing it soundly[2].
- **Replacing every `unwrap()`.** One on a value you just checked, in a test or at startup is fine. Fix the paths untrusted input reaches.
- **Ranking crates by how much `unsafe` they contain.** `bcrypt` forbids unsafe code yet panicked on input[11]. Ask what a crate parses and how it's tested.

What changed in 2026

- **Build-time malware hit a popular crate.** On 20 August, malicious `arrayref` 0.3.10, `internment` 0.8.7 and `append-only-vec` 0.1.9 were online for up to 107 minutes. Check `~/.cargo/registry/cache`[13].
- **Cargo gets cooldowns.** Rust 1.100, due 12 November, will add `registry.global-min-publish-age`, which skips newer versions not yet in `Cargo.lock`[27]. Set it in `.cargo/config.toml` to the supply-chain sheet's `"3 days"`.
- **Miri copied every environment variable into `target/`** until the 22 September nightly, exposing CI secrets to pull requests that read the cache[28]. Clear it and rotate them.

Mistakes that cost the most

- **Dropping hardening because the code is Rust.** Android's hardened allocator made its first Rust memory-safety bug, CVE-2025-48530, non-exploitable[2]. Keep hardened allocators, sandboxes and least privilege.
- **Giving build and test jobs secrets.** Neither Cargo nor Miri guarantees that environment variables stay out of `target/`[28]. Give secrets only to the deploy step.

What this looks like for your code

New project

- Start on edition 2024 with `unsafe_code = "forbid"` and the Clippy panic lints under `[workspace.lints]`; each crate must opt in.
- Set `overflow-checks = true` for release from the first commit.
- Make `cargo deny check` and `cargo clippy -- -D warnings` required CI checks.

Existing codebase

- List every `unsafe` block, `extern` block and `unsafe impl`, and review first those that touch parsed input.
- Add SAFETY comments as you go, then turn on `undocumented_unsafe_blocks`.
- Turn on release overflow checks and benchmark. Keep `wrapping_*` where wrapping is intended.

High-assurance or regulated

- Prove the `unsafe` core with Kani and fuzz every parser continuously.
- Use `cargo vet` so each dependency has a named reviewer, and build releases with `cargo auditable`.
- Ferrocene's compiler is qualified for ISO 26262 ASIL D, IEC 61508 SIL 3 and IEC 62304 Class C[29].

Version history

3 Nov 2026, version 1.0 Published after a copy edit and a source-by-source fact check, which corrected 2 fixes and narrowed 9 statements to what the sources say.

Sources

1. 2025 CWE Top 25 Most Dangerous Software Weaknesses. MITRE, updated 15 Dec 2025. cwe.mitre.org
2. Rust in Android: move fast and fix things. Google, 13 Nov 2025. blog.google
3. Rust Reference: Behavior not considered unsafe. Rust, checked 3 Nov 2026. doc.rust-lang.org
4. Cargo Book: The [lints] section. Rust, checked 3 Nov 2026. doc.rust-lang.org
5. Clippy Lints. Rust, checked 3 Nov 2026. clippy.rust-lang.org
6. Miri (README). Rust, checked 3 Nov 2026. github.com
7. CVE-2025-68260: rust_binder: fix race condition on death_list. Linux kernel, 16 Dec 2025. cve.org
8. Cargo Book: Profiles. Rust, checked 3 Nov 2026. doc.rust-lang.org
9. RUSTSEC-2026-0007: Integer overflow in BytesMut::reserve. RustSec, 3 Feb 2026. rustsec.org
10. Rust Reference: Panic. Rust, checked 3 Nov 2026. doc.rust-lang.org
11. RUSTSEC-2026-0199: Panic in bcrypt::verify on non-ASCII hash input. RustSec, 20 Jun 2026. rustsec.org
12. Rust Reference: Procedural macros. Rust, checked 3 Nov 2026. doc.rust-lang.org
13. Supply chain attack on arrayref. Rust Blog, 20 Aug 2026. blog.rust-lang.org
14. cargo-deny (README). Embark Studios, checked 3 Nov 2026. github.com
15. crates.io: development update. Rust Blog, 13 Jul 2026. blog.rust-lang.org
16. RUSTSEC-2026-0103: Use-After-Free and Double Free in IntoIter::drop When Element Drop Panics. RustSec, 14 Apr 2026. rustsec.org
17. RUSTSEC-2026-0317: A 512-byte workbook can provoke a multi-gigabyte allocation and abort the process. RustSec, 29 Sep 2026. rustsec.org
18. Security advisory for the standard library (CVE-2024-24576). Rust Blog, 9 Apr 2024. blog.rust-lang.org
19. Security advisory for the standard library (CVE-2024-43402). Rust Blog, 4 Sep 2024. blog.rust-lang.org
20. std::path::Path::join. Rust, checked 3 Nov 2026. doc.rust-lang.org
21. CVE-2026-33056: tar-rs: unpack_in can chmod arbitrary directories by following symlinks. GitHub, 20 Mar 2026. cve.org
22. Edition Guide: Unsafe extern blocks. Rust, checked 3 Nov 2026. doc.rust-lang.org
23. cargo audit (README). RustSec, checked 3 Nov 2026. github.com
24. rust-analyzer manual: Configuration. rust-analyzer, checked 3 Nov 2026. rust-analyzer.github.io
25. Kani Rust Verifier: Getting started. Kani project, checked 3 Nov 2026. model-checking.github.io
26. cargo vet book. Mozilla, checked 3 Nov 2026. mozilla.github.io
27. Cargo Book: Changelog. Rust, checked 3 Nov 2026. doc.rust-lang.org
28. GitHub Actions leaking secrets when Miri output is cached. Rust Blog, 21 Sep 2026. blog.rust-lang.org
29. Ferrocene. Ferrous Systems, checked 3 Nov 2026. ferrocene.dev